ORBIT ESTIMATION ALGORITHMS FOR A MICROSATELLITE RENDEZVOUS WITH A NON-COOPERATIVE TARGET

THESIS

John P. Heslin, Lieutenant Colonel, USAF

AFIT/GSS/ENY/05-M02

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the United States Air Force, Department of Defense, or the U.S. Government.

AFIT/GSS/ENY/05-M02

ORBIT ESTIMATION ALGORITHMS FOR A MICROSATELLITE RENDEZVOUS WITH A NON-COOPERATIVE TARGET

THESIS

Presented to the Faculty

Department of Aeronautics and Astronautics

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

In Partial Fulfillment of the Requirements for the

Degree of Master of Science in Astronautical Engineering

John P. Heslin, MS

Lieutenant Colonel, USAF

March 2005

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

AFIT/GSS/ENY/05-M02

ORBIT ESTIMATION ALGORITHMS FOR A MICROSATELLITE RENDEZVOUS WITH A NON-COOPERATIVE TARGET

John P. Heslin, MS

Lieutenant Colonel, USAF

Approved:

_____	_____
Dr. Richard G. Cobb (Chairman)	Date
_____	_____
Kerry Hicks, Lt Col, USAF (Member)	Date
_____	_____
Nate Titus, Lt Col, USAF (Member)	Date

Abstract

This study investigated the minimum requirements to establish a satellite tracking system architecture for a microsatellite to rendezvous with a non-cooperative target satellite. A prototype optical tracking system was reviewed with emphasis on a proposed tactical employment that could be used by technologically unsophisticated state or non-state adversaries. With the tracking system architecture selected, simulated tracking data was processed with a Non-Linear Least Squares batch orbit estimation algorithm and a Bayes sequential orbit determination filter in MATLAB to update the target satellite's state vector.

Acknowledgments

I would like to express my sincere appreciation to the United States Air Force for the opportunity to pursue this research. I also owe enormous gratitude to the many Air Force professionals who lent their expertise and time to this effort, especially my faculty advisor, Dr Cobb, as well as Dr Hicks, for their insight and guidance. In addition, special thanks go to Dr Fullerton from Air Force Research Labs for providing an overarching context to the space tracking system architecture model.

Thanks must also go to all of my classmates in both the Space Operations program as well as the Astronautical Engineering program for their camaraderie, perspective, and invaluable assistance with the day-to-day academics. In particular, Major Don Baker and Major Jim Pritchard epitomized excellence and friendship.

Finally, I am deeply indebted to my family for their prayers and encouragement throughout the effort.

John P. Heslin

Table of Contents

Page

List of Figures

Page

List of Tables

Page

ORBIT ESTIMATION ALGORITHMS FOR A MICROSATELLITE RENDEZVOUS WITH A NON-COOPERATIVE TARGET

I. Introduction

Background

Orbital rendezvous is an evolving engineering problem. From the early US space program, when the Apollo 9 Command Module first rendezvoused and re-docked with the Lunar Module in 1969, to current interplanetary reconnaissance, the engineering efforts have focused on optimizing either the time to rendezvous or the necessary energy. However, these efforts have all relied on highly accurate state knowledge of both the maneuvering satellite and the station satellite or celestial body. In rendezvous maneuvers between satellites, their initial and enroute states, $\vec{X} = \begin{bmatrix} \vec{r} & \vec{v} \end{bmatrix}_{6\times1}$, are typically equally well measured, if not identically measured, by ground controllers and known by at least the maneuvering satellite if not also by the station satellite to facilitate a merge. This can be called a cooperative rendezvous. In cases where the station satellite's state is not well known by either the ground station or the maneuvering satellite, it must be determined or estimated based on observation before a rendezvous can be attempted. This is known as a non-cooperative rendezvous, and the station satellite in this case can appropriately be referred to as a target. The intercept and rendezvous of a target satellite will be referred to in this paper as the *targeting problem*.

The key to the targeting problem is knowing the target's state well enough to execute a rendezvous maneuver to within the interceptor's terminal sensors. In this context, satellite state estimation is a qualitative problem. A satellite's state is a six-

element vector, thus in order to establish an initial orbit determination, observations must continue to be made until six elements of data are measured. For example, typical ground-based radars will directly measure three elements of data such as a target's range, azimuth, and elevation, implying two observations must be made before an orbit can be determined. For optical telescopes, only a target's azimuth and elevation can be directly measured, thus three observations must be made. In both cases, measurement errors affect the accuracy of the initial orbit determination. These errors, combined with the uncertainty of orbit perturbations typical of every practical satellite, result in rapidly declining confidence of the target's actual state when the initial orbit determination is computationally propagated into the future.

Knowledge of a target's future state depends first on accurately estimating its current state and second on accurately modeling the dynamics of its orbit, maneuvers notwithstanding. The standard deviation in an estimate, $\sigma_{\bar{x}}$, is inversely related to the number of observations made, so a state estimate's accuracy is improved with additional observations made of the target. Per Wiesel [18, pg 22, 23]:

$$\sigma_{\bar{x}} = \frac{\sigma_i}{\sqrt{N}} \tag{1.1}$$

where *N* is the number of measurements, σ_i is the expected standard deviation instrument error for measurement *i*, and $\sigma_{\bar{x}}$ is the standard deviation associated with the estimated value. Thus,

$$\sigma_{\bar{x}}^2 = \frac{1}{\sum_{i=1}^{N} 1/\sigma_i^2} \tag{1.2}$$

Implying the best estimate of the state, $\vec{x}$, will be

$$\vec{x} = \sigma_{\vec{x}}^2 \sum_{i=1}^{N} \frac{x_i}{\sigma_i^2} \tag{1.3}$$

The estimate is a function of the accuracy of the data and the amount of data collected. With batch algorithms such as Non-linear Least Squares, all of the data are processed at once in order to establish an estimate of the state at the epoch of the first observation. As new data become available, the entire set of data must be reprocessed as a new batch. From Equation (1.3) above, additional data will improve the estimate, but the estimate will still be of the target's state at the epoch of the first observation. Sequential estimators on the other hand, such as Bayes, allow new data to be processed treating the initial estimate as its own data point with associated covariance. In this manner, the epoch of the estimate can be brought forward in time with increased accuracy.

Problem Statement

This research extends an Air Force Institute of Technology Department of Aeronautics and Astronautics (AFIT/ENY) effort begun in 2003 to model space tracking system architectures in the MATLAB programming language to answer two key questions. First, can a non-cooperative target's state be estimated well enough from data gathered by a network of optical sensor sites to rendezvous a microsatellite interceptor close enough for its terminal sensors to find and track the target to a merge? Second, can the microsatellite be guided to the merge within its delta-v budget? The problem is approached from the perspective of a space user that lacks the vast space surveillance

radar infrastructure of the United States, European Union, or the Former Soviet Union necessary to rapidly and accurately ascertain a satellite's orbit. The hypothetical space user may be in active daily contact with their maneuvering satellite, but not the target. The objective of this research has been to design a functioning Bayes algorithm in MATLAB for low-earth-orbit (LEO) target satellite tracking by a network that includes one space surveillance radar and several other ground sites employing optical sensors, and analyze performance. For the work herein, the code was tested against simulated observations of Space Shuttle mission STS-109, a representative LEO target. The code design is based on the work of Dr William Wiesel and Brian Foster, MS. In addition, Dr Wiesel's discussion of a pseudo-observation and its associated dynamic covariance was essential to overcoming difficulties intrinsic to data residuals for LEO satellites. Finally, and importantly, the results must be understood in the context of the purpose for the research. Thus the question to be answered is "Given one radar site and a set of optical sensors (such as Boeing-Rocketdyne's 58-lb prototype portable optical satellite tracker[*] [9]), can LEO target tracking be accomplished well enough for counter-space rendezvous mission planning?"

Methodology

There are several methods to find the estimate of target state $\bar{x}$ based on N observations. Previous AFIT research on the targeting problem, for example, examined the efficacy of processing N observations as a batch using a non-linear least squares filter

[*] Specifications included in Appendix A

algorithm. Studies found that 20 consecutive radar observations taken at 60 second intervals of a target in an unperturbed 830-km altitude sun-synchronous orbit would yield an estimate of the target's initial position state accurate to within 184 meters, however, the simulation did not evaluate velocity errors [3, pg 54]. Although an orbital merge to within 184 meters of a target could be considered a successful rendezvous, that estimate is for the target's state at a specific epoch, usually the time of the initial or final observation.

Thus, at the moment the last observation is made, the accuracy of the estimate (assuming it was processed and calculated at the instant the last observation was made) will have degraded for 20 minutes in accordance with the variations in the dynamics of the target's orbit at a rate proportional to the uncertainty of the state estimate. In addition, a co-planar rendezvous maneuver initiated from a parking orbit of, say 185-km to the target altitude of 830-km could take anywhere from 30 minutes to perhaps 3 hours [2, pg146]. In the study noted above, the target satellite's orbital period was roughly 100 minutes.

The growth of uncertainty in an estimated state is illustrated by Wiesel. Consider the target's state as described by the six classical orbital elements $a, e, i, \Omega, \omega, M$ and the associated dynamics that of the classical two body problem. In this case, only the mean anomaly M changes with time:

$$M = M_o + \sqrt{\frac{\mu}{a_o^3}}(t - t_o) \tag{1.4}$$

Its associated variance σ_M^2 grows as a function of gravitational parameter μ, semi-major axis a_o, and the variance of the semi-major axis, σ_a^2:

$$\sigma_M^2(t) = \sigma_M^2(t_o) + \frac{9}{4}\mu a_o^{-5}(t - t_o)^2 \sigma_a^2 \tag{1.5}$$

For the sun-synchronous target described above, assume for the sake of discussion that σ_a = 32 m and $\sigma_M(t_o)$ = 0.001 degrees. Three hours after the estimate's epoch, $\sigma_M(3hours)$ will be approximately 0.002 degrees. Although small, σ_M will continue to grow from the epoch of the non-linear least squares estimate. Even if data are later added to the batch, and the entire data set reprocessed, the statistical confidence in the estimate will continue to degrade with time measured from the epoch of the estimate.

Sequential filter algorithms allow new data to be processed after an initial estimate has been established, and allow the epoch for the new estimate including all the information from the old estimate (if it is still useful) to be brought forward to the epoch of the new set of data. For fundamentally deterministic systems such as an orbiting satellite, the Bayes sequential filter estimation algorithm allows new observations of a target from a ground site to be processed in this manner, and fit to the known dynamics of the target.

Preview

This research effort produced mixed results in ascertaining a high-confidence state estimate for a realistic LEO target, in this case the Space Shuttle, with the sequential filter and improved dynamics added to the AFIT/ENY space tracking system architecture

model. A sequential algorithm was successfully coded that converged on an estimate given short-arc dense data sets. However, the estimated state did not accurately reflect the actual state of the target, making a prediction of a future state problematic.

II. Literature Review

Chapter Overview

In this chapter, sources for the Non-Linear Least Squares and Bayes sequential filtering algorithms explored in this study are reviewed, as well as current research into variations in the algorithms. The targeting problem as viewed by Chobotov, Vallado, and Wiesel is briefly discussed to provide perspective on the significance of this study. The relevance of research into non-cooperative target state estimation for rendezvous mission planning from the military perspective is explained in detail by Foster, and is looked at in overview below to provide context.

Orbit Estimation

Although orbit estimation as a process has been documented and refined over the past 50 years, increasing computing power has allowed researchers to explore techniques for establishing and maintaining accurate estimates using sequential algorithms that can process data representing non-deterministic dynamics such as that of maneuvering targets. Although the algorithms for batch and sequential data filtering are published in texts such as Wiesel's "Modern Methods of Orbit Determination" and Vallado's "Fundamentals of Astrodynamics and Applications," their implementation to process any given set of orbit observations does not guarantee convergence to an estimate, nor precision in that estimate, which depend on the quantity of data being processed, as well as degree to which the data are deterministic. The focus of current orbit estimation

literature is either on improvements to the filter algorithms to accommodate small data sets or increasingly non-deterministic orbital dynamics.

The two primary references for discussion of orbit estimation are Wiesel's "Modern Methods of Orbit Determination" and Vallado's "Fundamentals of Astrodynamics and Applications." Wiesel's text is essential to understanding the estimation problem as well as the batch and sequential approaches to estimating. He outlines the process for each filter type (reproduced in Chapter III), and provides insight to the specific of the observation and covariance matrices. For example, Wiesel goes into explicit detail on the derivation of the observation relation matrices.

Vallado's text provides practical pseudo code for batch and Bayes sequential filters, but relies heavily on approaches that provide approximations, versus the more rigorously derived approaches analyzed by Wiesel. For example, whereas Wiesel explicitly derives the observation relation matrix H in order to form the observability matrix T from the product $H\Phi$, Vallado approximates the observability matrix directly from a gradient established by finite differencing.

Vallado and Scott Carter describe orbit determination from short-arc observational data in reference [16]. The study concluded that accurate orbit determination from short arcs of observations depends on the availability of "dense" observational data, defined as data taken at 1 second intervals for at least 6 minutes. This amount of data ensures sufficiency to obtain an accurate estimate and covariance matrix. Interestingly, Vallado found that only moderate data are required to maintain the orbit estimate once the initial state vector is properly formed (as long as the target doesn't

maneuver in the meantime), meaning that if observations can be made only once a day, or even once a week after the orbit is established, a target's state can be known with an accuracy on the same order as that had when data was densely collected for the estimate [16, pg1597].

Felix Hoots [5] discusses measurement biases (range, angle) on the covariance of an orbit estimate and subsequent predictions. He demonstrates that uncompensated measurement bias does not affect the covariance explicitly, although it influences the covariance through the amount by which it skews the solution.

Soohong Kim and Joohwan Chun [7] compare Bayesian bootstrap filtering to extended Kalman filtering for LEO satellite orbit determination. In the Bayesian bootstrap approach, the orbital dynamics to be used to generate the predicted orbit state is a discrete-time version of the continuous-time approach. The state probability distribution function (PDF) is represented as a set of random samples of the data distribution and the state distribution. The likelihood of the propagated state is then compared to the distribution of the predicted PDF. The approach used by Kim and Chun is applied to large set of data gathered from magnetometer sensors onboard the satellites, and doesn't lend itself to short-arc data sets inherent to ground-based LEO orbit estimation. Similarly, Sangwoo Cho and Joohwan Chun [1] evaluate Bayesian bootstrap filtering for multiple mobile position determination using LEO satellites.

Donald Hoffman and Richard Hujsak describe real time orbit determination with sequential filters in reference [4]. They evaluated the performance of a commercial real-time orbit determination (RTOD) filter based on the Kalman filter. The RTOD filter

accepts real-time satellite tracking observations and, within seconds of data receipt, reports updated estimates of orbital parameters. They describe batch-least-squares curve fitting techniques as being based on residuals formed from the differences between measured and predicted observations. The estimated state consists of the set of parameters from which the best predicted observations can be generated. In turn, the best predicted observations are those that minimize the sum of the squares of all the observation residuals. A batch least-squares estimate is a function of the most recent batch of observations only, and the estimated state is not a function of any previous state estimates. According to Hoffman, sequential filters have an advantage when there are time-varying uncertainties in the Orbit Determination problem. An example of such uncertainties is force model truncation. Each simplification in the force model such as low order oblate earth approximations and spherical satellite drag approximations introduces an error into the orbit determination process that is dynamically correlated and time-varying.

There is very little written on networking sensors for analysis of simultaneous observations. A paper by Walt Truszkowski, reference [13], describes the prototype development of automated ground-based orbit determination function and evaluates its performance in providing current orbit estimates of the SOHO satellite to the SOHO control center via a web interface. In this case, data are being collected by remote observer locations (automatons, specifically Artificial Intelligence (AI) java applets), and provided via a web-connected network to applications that process the observation data and provide continuously updated orbit state estimates to NASA users.

Richard Tansey [9] describes experiments with a 58-lb portable LEO satellite tracker. His effort grew out of a desire to develop a small, inexpensive, portable tracking system that featured a 12" Schmidt Cassegrain telescope and mount that would require only one operator to run. The system developed and tested has dimensions of 21"x15"x10", and uses an integral microcontroller and 9.5" worm gears to achieve adjustable slew rates of 6 degrees per second. With a CCD and off-the-shelf tracking software, LEO pointing is demonstrated on a f/10 eight inch telescope to less than 8 μrad for periods of 10 seconds, and 50 μrad (0.00286deg) for entire LEO orbit passes. A closed loop one Hz video tracker is also described with automatic tracking of mag 7/8 satellites using a 12" telescope. Requiring only one operator, this system can be transported by a small car and be operational in a few hours. Software, equipment, and instrument accuracies are reproduced from Tansey's report in Appendix A.

The Targeting Problem

The targeting problem has been addressed by Chobotov [2] and Vallado as well as Wiesel. However, in all treatments, the state of both satellites is assumed to be known throughout the rendezvous. For example, Vallado [15] treats the targeting problem as Lambert's problem for either an intercept or a rendezvous. The solution requires perfect knowledge of the targets state. Rendezvous maneuvering and satellite formations are the focus of recent USAF and commercial experiments. AFRL XSS-10 and XSS-11 experiments explore proximity maneuvers on orbit. Orbital Sciences Corp is launching the Demonstration of Autonomous Rendezvous Technology (DART) experiment to evaluate the performance of the Advanced Video Guidance Sensor (AVGS). The AVGS

is a terminal sensor that requires reflectors on the target to ascertain relative position for proximity maneuvers. The DART experiment focuses on the merge, as opposed to assessing the orbit of a target.

Counter-space motivations

Concern about Chinese research into rendezvous via nano-satellites is explored in detail by Foster. AFIT research including this study is specifically motivated by a desire to assess the practical counter space threat posed by the mission they propose.

Summary

Active research into orbit estimation and the targeting problem is ongoing because of the centrality of these concepts to space operations. A review of the current literature shows that orbit estimation is an ongoing engineering problem, while non-cooperative rendezvous is only beginning to be broached in the last few years, motivated by USAF interest in protecting US and national on-orbit space assets. The targeting problem in particular has not been found to be evaluated from the perspective of optimizing a rendezvous with an imperfectly known target state.

III. Methodology

Chapter Overview

Orbit determination generally consists of two major processes: the process of determining an initial orbit for a satellite, referred to as initial orbit determination, and the process of taking observations and forming an updated state vector for a satellite, known as estimation. Previous work done by Foster [3] on the AFIT/ENY space tracking system architecture model looked at orbit determination from an on-orbit sensor. This study focuses on the ground-based orbit estimation problem. The approach to this orbit determination problem is derived from the operational perspective of the question being asked. The Non-Linear Least Squares batch and Bayes sequential algorithms are reviewed, as are discussions of the difficulties of these algorithms for low earth orbit targets, and the changes made to these algorithms to accommodate the LEO target dynamics and orbit estimation problem. A key change to the algorithms is the use of a correction scale factor based on Vallado's finite differencing technique. In addition, because the estimation techniques include the use of Wiesel's pseudo-state vector concept, and because that concept pre-supposes that the target's range is being measured from the ground site, a specific optical ground site architecture is proposed and discussed.

Approach

The question to be answered is "Given one radar site and a set of optical sensors (such as Boeing-Rocketdyne's 58-lb prototype portable optical satellite tracker), can LEO target tracking be accomplished well enough for counter-space operations (e.g.

rendezvous)?" Following previous AFIT research by Brian Foster, who used a non-linear least squares algorithm programmed in MATLAB to estimate a low-earth-orbit target satellite state, this project's approach was to develop a sequential filter to improve estimates over time. This work is based on the Bayes algorithm in MATLAB modeled after Fortran code written by Dr William Wiesel to estimate interplanetary trajectories.

Non Linear Least Squares Algorithm

The Bayes filter code is based on equations, algorithms, and pseudo-code found in Wiesel and Vallado. Because the Bayes algorithm is a modification of non-linear least squares, it is important to first understand the non-linear least squares algorithm. Nonlinear Least Squares Algorithm and Pseudo-code from Wiesel [17, pg 79], and Vallado [14, pg 712]:

For each observation time t_i:

I. Propagate the state vector to the observation time t_i,

Also, obtain the state transition matrix $\Phi(t_i, t_o)$, where t_o is the time at epoch.

II. Obtain the residual vector $\vec{r}_i = \vec{z}_i - G(\vec{X})$, where *G(X)* is the observation relationship function to relate the propagated state to the measurements *z*.

Calculate H_i for this particular data point, equal to the partial of *G* with respect to the state.

Calculate $T_i = H_i \Phi$

III. Add new terms to the running sums of the matrix

$$\sum_i T_i^T Q_i^{-1} T_i$$

Where Q is the instrument covariance matrix, and add new terms to the vector

$$\sum_i T_i^T Q_i^{-1} r_i$$

When all data has been processed:

IV. Calculate the covariance of the correction

$$P_{\delta \bar{X}} = \left(\sum_i T_i^T Q_i^{-1} T_i \right)^{-1} \tag{3.1}$$

Calculate the state correction vector at epoch

$$\delta \vec{X}(t_o) = P_{\delta \bar{X}} \sum_i T_i^T Q_i^{-1} r_i \tag{3.2}$$

V. Correct the reference trajectory

$$\vec{X}_{ref+1}(t_o) = \vec{X}_{ref}(t_o) + \delta \vec{X}(t_o)$$

VI. Determine if the process has converged. If it has not, re-iterate I-V. If the process has converged, $\vec{X}_{ref+1}(t_o)$ is an estimate of the orbit state at epoch with covariance $P_{\bar{X}}$

Finally, check the residuals to identify any secular or quadratic trends that may indicate a change in the target's dynamics.

<u>Non-Linear Least Squares Pseudo-code</u> (based on Vallado [15] algorithm 62, "Differential Correction")

Begin loop to process each observation

FOR i = 1 to the number of observations

Propagate the nominal state $\vec{X}_{ref}$ each t_i and find computed observations:

$$\vec{X}_{ref_i} = \int_{t_o}^{t} \dot{\vec{X}}_{ref} dt + \vec{X}_{ref}$$

Run Vallado Routine *RAZEL*:

inputs: $\vec{r}_{ECI}, \vec{v}_{ECI}, site\ elevation, site\ latitude, site\ LST$

outputs: $range, azimuth, elevation, range\ rate$

Run Wiesel Routine *OBSER* to create an expected data vector from the propagated reference orbit (*computed data*)[†], calculate observation relation matrix *H*, and instrument covariance matrix *Q*.

Find the residual vector $\vec{\mathrm{r}}$ as observed minus computed data for each t_i

Integrate the equations of motion, perturbations and variations to find Φ:

$$\begin{cases} \dfrac{d\vec{X}}{dt} = \ddot{\vec{X}}_{2-body} + \ddot{\vec{X}}_{nonspherical} + \ddot{\vec{X}}_{drag} + \ddot{\vec{X}}_{3-body} \\ \vec{A} = \dfrac{\partial \dot{\vec{X}}}{\partial \vec{X}} \qquad \Phi(t,t_o) = \int_{t_o}^{t} \vec{A}(t)\Phi(t,t_o)dt \end{cases}$$

The Φ matrix at t_o is the identity matrix as provided to the propagator, and the matrix A in the propagator is comprised of the equations of perturbation and variation of the equations of motion.

Equations of Motion:

The target satellite's motion is governed by the 2-Body equations of motion such that:

$$\dot{\vec{r}} = \vec{v} \tag{3.3}$$

$$\ddot{\vec{r}} = -\frac{\mu}{r^2}\frac{\vec{r}}{r} \tag{3.4}$$

[†] OBSER routine is included in Appendix G

with 2-Body acceleration perturbed by forces related to atmospheric drag, earth's oblateness, and 3rd body accelerations due to the Sun and Moon such that

$$\ddot{\vec{r}} = -\frac{\mu}{r^2}\frac{\vec{r}}{r} + a_p \tag{3.5}$$

Equations of Perturbation:

Oblate Earth perturbation [14, pg 554, Eq.8-48]

$$a_{J2} = \begin{bmatrix} \frac{-3J_2\mu R_\oplus^2 r_I}{2r^5}\left(1-\frac{5r_K^2}{r^2}\right) \\ \frac{-3J_2\mu R_\oplus^2 r_J}{2r^5}\left(1-\frac{5r_K^2}{r^2}\right) \\ \frac{-3J_2\mu R_\oplus^2 r_K}{2r^5}\left(3-\frac{5r_K^2}{r^2}\right) \end{bmatrix} \tag{3.6}$$

Drag perturbation [14, pp 525-526, Eq.8-28]

$$a_{drag} = -\frac{1}{2}\rho\frac{C_D A}{m}v_{rel}^2\frac{\vec{v}_{rel}}{|\vec{v}_{rel}|} \tag{3.7}$$

$$\vec{v}_{rel} = \frac{d\vec{r}}{dt} - \vec{\omega}_\oplus \times \vec{r} = \begin{bmatrix} \frac{dx}{dt} + \vec{\omega}_\oplus r_J \\ \frac{dy}{dt} - \vec{\omega}_\oplus r_I \\ \frac{dz}{dt} \end{bmatrix} = \begin{bmatrix} v_I + \vec{\omega}_\oplus r_J \\ v_J - \vec{\omega}_\oplus r_I \\ v_K \end{bmatrix} \tag{3.8}$$

Exponential model [14, pg 535, Eq.8-34]

$$\rho = \rho_o e^{\frac{-(h_{ellp}-h_o)}{H}} \tag{3.9}$$

3rd body perturbations (sun, moon)

$$a_{3-Body} = \mu_{3rdBody}\left(\frac{\vec{r}_{3rdBody\,to\,Sat}}{r^3_{3rdBody\,to\,Sat}} - \frac{\vec{r}_{3rdBody\,to\,Earth}}{r^3_{3rdBody\,to\,Earth}}\right) \tag{3.10}$$

The equations of variation have been included in Appendix B. The equations of variation form the Jacobian matrix [A] to find $\dot{\Phi}(t)$. The elements of A, A(i,j) are the partial derivatives of the ith equation of motion with respect to the jth element of the state vector per equations (3.11) and (3.12) below:

$$\frac{d}{dt}\vec{X} = f_{2-Body}(X) = \begin{pmatrix} \vec{v} \\ \dfrac{-\mu\vec{r}}{r^3} \end{pmatrix}_{6\times 1} \tag{3.11}$$

$$\frac{d}{dt}\Phi(t,t_o) = \dot{\Phi} = A(t)\Phi(t,t_o) \tag{3.12}$$

Calculate G matrix. G is the predicted data vector as a function of the current state, referred to as the observation relationship, and depends on the type of data that was collected to be processed by the filter. For example, if one data element were the range magnitude from a radar observation, that element of the G vector would be:

$$G_i = (\vec{X}) = range, \rho = \sqrt{x^2 + y^2 + z^2} \tag{3.13}$$

Calculate the H matrix. H is the linear observations model, an nx6 matrix of partial derivatives of G that relates the data to the state evaluated on the reference trajectory. Thus H is given as:

$$H_{ij}\Big|_{i \text{ corresponds to } \rho \text{ data element}} = \frac{\partial G_{i\Rightarrow\rho}}{\partial \vec{X}_j}\Big|_{X_{ref}} = \begin{bmatrix} \frac{x}{\sqrt{x^2+y^2+z^2}} \\ \frac{y}{\sqrt{x^2+y^2+z^2}} \\ \frac{z}{\sqrt{x^2+y^2+z^2}} \\ 0 \\ 0 \\ 0 \end{bmatrix}^T = \begin{bmatrix} \frac{x}{\rho} & \frac{y}{\rho} & \frac{z}{\rho} & 0 & 0 & 0 \end{bmatrix} \tag{3.14}$$

With H, find $\vec{T}_i = H\Phi$ (3.15)

$$Q = \begin{bmatrix} \sigma^2_{measurement\ 1} & 0 & 0 \\ 0 & \sigma^2_{measurement\ 2} & 0 \\ 0 & 0 & \sigma^2_{measurement\ n} \end{bmatrix} \tag{3.16}$$

Accumulate $T^T Q^{-1} T$ and $T^T Q^{-1} \vec{r}_i$ where $\vec{r}_i$ is the residual vector.

END LOOP

$$\delta \vec{X} = (T^T Q^{-1} T)^{-1} T^T Q^{-1} \vec{r} = P T^T Q^{-1} \vec{r} \tag{3.17}$$

Check for convergence. Update the state vector if not converged: $\vec{X}_{ref+1} = \vec{X}_{ref} + \delta \vec{X}$

If the process has converged: $\vec{X}_{Estimate} = \vec{X}_{ref}$

Bayes Algorithm

The Bayes Algorithm and Pseudo-code are similar to that of non-linear least squares. The Bayes algorithm and Pseudo-code from Wiesel [17, pg 109], and Vallado, [14, pg 715] follows:

I. Bring the old estimate and its covariance to the new epoch

Call the epoch of the original estimate t_{o-} and the epoch of the new data set t_o

$$\vec{X}((-),t_o) = \Phi(t_o,t_{o-})\vec{X}_{o-}(t_{o-}) \tag{3.18}$$

$$P((-),t_o) = \Phi(t_o,t_{o-})P(t_{o-})\Phi^T(t_o,t_{o-}) \tag{3.19}$$

Pick $\vec{X}_{ref} = \vec{X}(-)$

With N new observations, for each observation time t_i:

II. Propagate $\vec{X}_{ref}$ and Φ to the observation time t_i,

Calculate $\vec{r}_i = \vec{z}_i - G(\vec{X})$, Calculate H_i, Calculate T_i

(any residual rejection occurs at this point)

III. Add new terms to the running sums of the matrix

$$\sum_i T_i^T Q_i^{-1} T_i$$

Add new terms to the vector

$$\sum_i T_i^T Q_i^{-1} r_i$$

When all data have been processed:

IV. Calculate the new covariance of the state correction

$$P^{-1}(+) = P^{-1}(-) + \sum_{i=1}^{N} T_i^T Q_i^{-1} T_i \tag{3.20}$$

Calculate the estimate of the state correction

$$\delta\vec{X}(t_o) = P(+)(P^{-1}(-)(\vec{X}(-) - \vec{X}_{ref}) + \sum_{i=1}^{N} T_i^T Q_i^{-1} r_{zi} \tag{3.21}$$

V. Correct the reference trajectory

$$\vec{X}_{ref+1}(t_o) = \vec{X}_{ref}(t_o) + \delta\vec{X}(t_o) \tag{3.22}$$

VI. Determine if the process has converged. If it has not, re-iterate II-V. If the process has converged, $\vec{X}_{ref+1}(t_o)$ is an estimate of the orbit state at epoch with covariance $P(+)$

Finally, check the residuals to identify any secular or quadratic trends that may indicate a change in the target's dynamics. Per Wiesel, it is desirable to perform at least two iterations of this algorithm, since with the initial $\vec{X}_{ref} = \vec{X}(-)$ the state residual is zero on the first iteration, and the previous state does not influence the new estimate until the second iteration. Also, the data residuals should be compared to the data covariance *Q*, and the state residual should be compared to $P(-)$. If the change is very large[‡], the previous estimate was probably not good enough to justify the use of the Bayes filter.

Bayes Pseudo-code

Bring the old estimate and its covariance to the new epoch

Call the epoch of the original estimate t_{o-} and the epoch of the new data set t_o

$\vec{X}((-),t_o) = \Phi(t_o,t_{o-})\vec{X}_{o-}(t_{o-})$

$P((-),t_o) = \Phi(t_o,t_{o-})P(t_{o-})\Phi^T(t_o,t_{o-})$

Pick $\vec{X}_{ref} = \vec{X}(-)$

Begin loop to process each new observation

FOR *i* = 1 to the number of observations

Propagate the nominal state $\vec{X}_{ref}$ each t_i and find computed observations:

$$\vec{X}_{ref_i} = \int_{t_o}^{t} \dot{\vec{X}}_{ref}\, dt + \vec{X}_{ref}$$

Run Vallado Routine *RAZEL*:

inputs: $\vec{r}_{ECI}, \vec{v}_{ECI}, site\,elevation, site\ latitude, site\ LST$

outputs: $range, azimuth, elevation, range\ rate$

Run Wiesel Routine *OBSER* to create an expected data vector from the propagated reference orbit (*computed data*), calculate observation relation matrix *H*, and instrument covariance matrix *Q*.

Find the residual vector $\vec{r}$ as observed minus computed data for each t_i

Integrate the equations of motion, perturbations and variations to find Φ:

$$\begin{cases} \dfrac{d\vec{X}}{dt} = \dot{\vec{X}}_{2-body} + \dot{\vec{X}}_{nonspherical} + \dot{\vec{X}}_{drag} + \dot{\vec{X}}_{3-body} \\ \vec{A} = \dfrac{\partial \dot{\vec{X}}}{\partial \vec{X}} \qquad \Phi(t,t_o) = \displaystyle\int_{t_o}^{t} \vec{A}(t)\Phi(t,t_o)dt \end{cases}$$

The propagator used by the Bayes algorithm is the same as that used by the Non-Linear Least Squares code, thus Φ and A are described in the section above.

Calculate G matrix. G is the predicted data vector as a function of the current state, referred to as the observation relationship. For example, if one data element were the range magnitude from a radar observation, that element of the G vector would be:

$$G_i = (\vec{X}) = range, \rho = \sqrt{x^2 + y^2 + z^2} \tag{3.23}$$

---

‡ Wiesel [17, pg 110]

Calculate the H matrix. H is the linear observations model, an nx6 matrix of partial derivatives of G that relates the data to the state evaluated on the reference trajectory. Thus H is:

$$H_{ij}\Big|_{i \text{ corresponds to } \rho \text{ data element}} = \frac{\partial G_{i\Rightarrow\rho}}{\partial \vec{X}_j}\Big|_{X_{ref}} = \begin{bmatrix} \frac{x}{\sqrt{x^2+y^2+z^2}} \\ \frac{y}{\sqrt{x^2+y^2+z^2}} \\ \frac{z}{\sqrt{x^2+y^2+z^2}} \\ 0 \\ 0 \\ 0 \end{bmatrix}^T = \begin{bmatrix} \frac{x}{\rho} & \frac{y}{\rho} & \frac{z}{\rho} & 0 & 0 & 0 \end{bmatrix} \tag{3.24}$$

$$\vec{T}_i = H\Phi$$

$$Q = \begin{bmatrix} \sigma^2_{measurement\ 1} & 0 & 0 \\ 0 & \sigma^2_{measurement\ 2} & 0 \\ 0 & 0 & \sigma^2_{measurement\ n} \end{bmatrix} \tag{3.25}$$

Accumulate $T^T Q^{-1} T$ and $T^T Q^{-1} \vec{r}_i$ where $\vec{r}_i$ is the residual vector.

END LOOP

$$P^{-1}(+) = P^{-1}(-) + \sum_{i=1}^{N} T_i^T Q_i^{-1} T_i \tag{3.26}$$

$$\delta \vec{X}(t_o) = P(+)(P^{-1}(-)(\vec{X}(-) - \vec{X}_{ref}) + \sum_{i=1}^{N} T_i^T Q_i^{-1} r_{zi}) \tag{3.27}$$

Check for convergence. Update the state vector if not converged: $\vec{X}_{ref+1} = \vec{X}_{ref} + \delta \vec{X}$

If the process has converged: $\vec{X}_{Estimate} = \vec{X}_{ref}$

Truth Model

The filter's performance was evaluated against realistic data collected by 13 hypothetical ground sites tracking Space Shuttle mission STS-109. Archived NORAD Two-Line Element Sets (TLEs) for STS-109 were downloaded from the Celestrak website [6] into the Satellite Orbit Analysis Program (SOAP), an orbit visualization application, to provide truth data.

According to the SOAP user manual, the SOAP application is an interactive simulation that employs three-dimensional animation to display the relative motions of the solar system, spacecraft, ground stations, aircraft, and surface vehicles. The positions and velocities of moving platforms are calculated from user-defined initial conditions using embedded propagation algorithms. For satellites, the initial conditions from files containing TLEs are inputs for the SGP (Simplified General Perturbations) family of propagators developed by agencies of the Air Force Space Command. SOAP uses version 3.0 of the NORAD SGP, SGP4, and SDP4 propagators. Ground Station Platforms represent positions defined relative to a rotating body frame such as the Earth. Inputs are latitude, longitude, and altitude. The standard SOAP definitions of these terms apply, with altitude being considered above earth mean sea level.

Realistic ground site locations were chosen based on whether the host country had trade relations with China (the hypothetical adversary), proximity to tropical latitudes, and proximity to increased ground elevation. Based on these criteria, the following observer ground sites were loaded into SOAP:

Table 1 Ground site locations

Place	latitude	longitude	elev (m)	Comments
China	28.2	102.0	2347	Chinese space track radar
Panama	8.5	-82.1	311	China is an import partner
Mali	16.4	2.3	346	China is an export partner
Niger	18.0	9.1	1011	China is an import partner
Gabon	-1.6	12.1	868	China is an export partner
Kenya	0.1	36.4	2329	China is an import partner
Indonesia 1	-1.4	101.3	300	China is a trade partner
Indonesia 2	1.6	115.2	511	China is a trade partner
Indonesia 3	-3.2	137.7	302	China is a trade partner
Peru	-6.2	-78.1	2998	China is an export partner
Columbia	0.7	-72.8	634	China is an import partner
Brazil 1	-14.0	-58.8	584	China is a trade partner
Brazil 2	-4.6	-39.9	325	China is a trade partner

An overview of the observer ground site locations is shown in Figure 1 below:

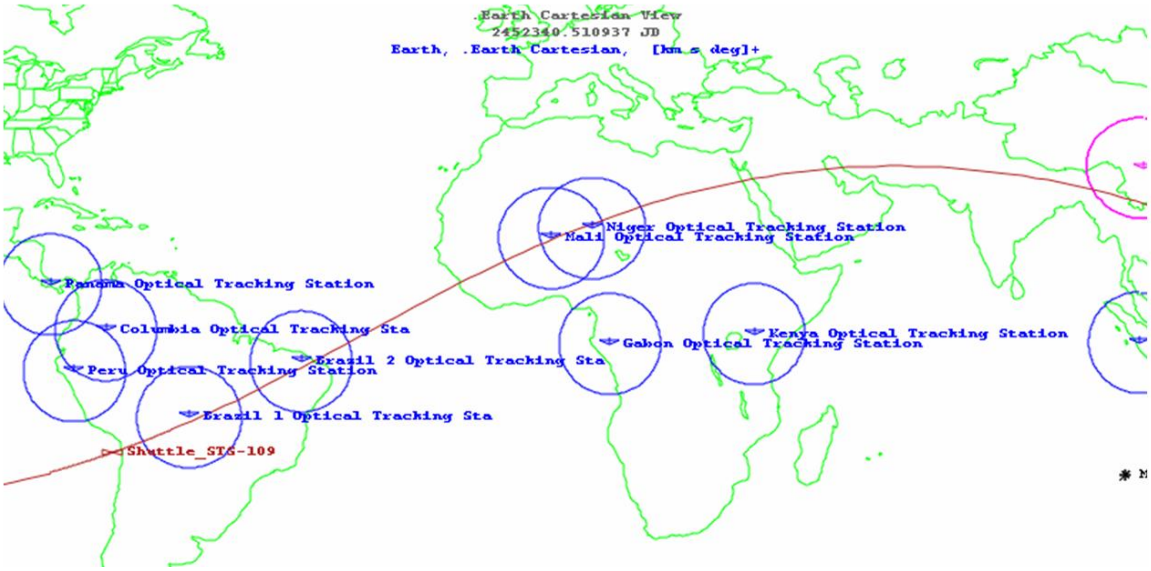

Figure 1 Ground site location overview

An example of the truth data display screen is shown in Figure 2 below:

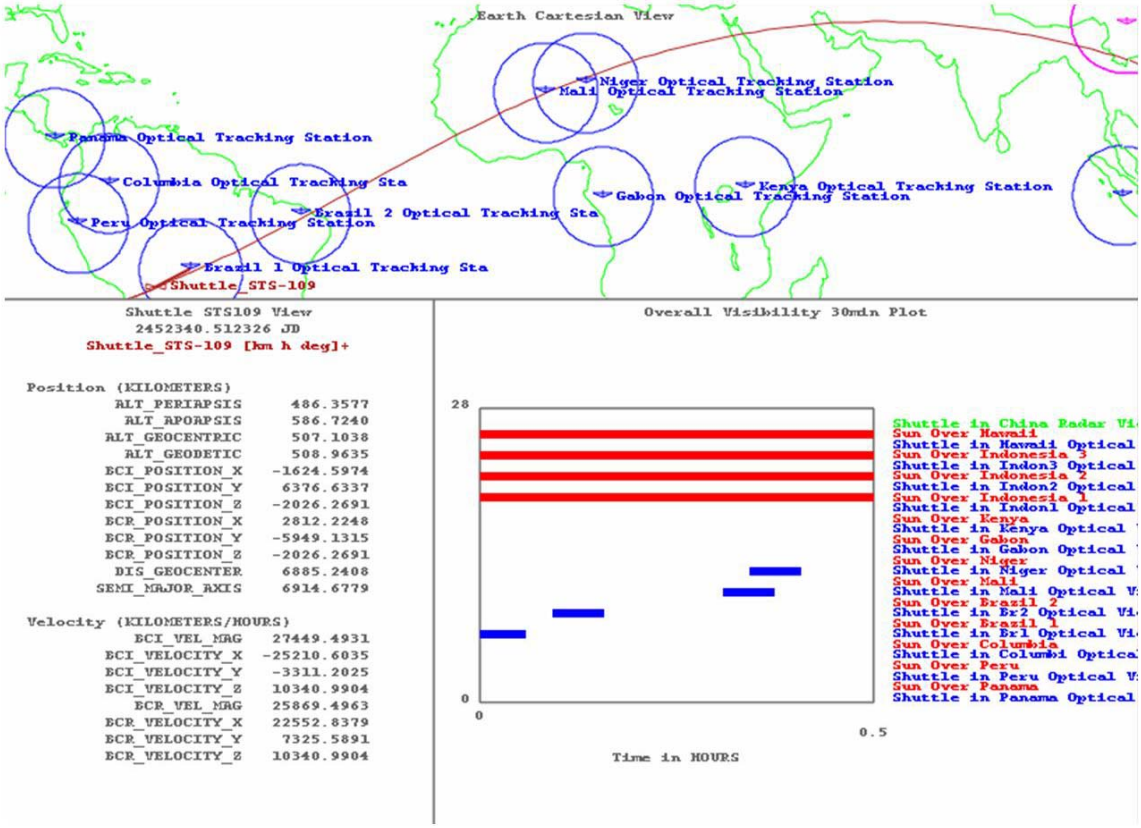

Figure 2 SOAP display for target observation and truth data

Algorithm Performance with Range/Azimuth/Elevation Data

In evaluating short dense arcs of radar data (range, azimuth, elevation) with the Non-Linear Least Squares algorithm coded in MATLAB, the estimator was seen to diverge rapidly. Performance troubleshooting revealed the filter would converge for LEO initial states and realistic noise levels, for several different data types (range and range-rate for example), but would not converge when the data residual vector included angle data residuals. Investigation of this phenomenon led to the realization that the LEO satellite state estimation problem is unlikely to converge when the residual vector

included azimuth and elevation data residuals, because of the behavior of those residuals during an iteration, and the result of those residuals on the subsequent state correction.

Essentially, when angle information is included in the residual vector, large persistent residuals in the calculated (propagated) orbit track would force the product $T^T Q^{-1} \bar{r}_i$ to be large (say 20% of the measurement range), even if the calculated position was off by only tens of kilometers (1% actual error). Figure 3 below illustrates the phenomenon.

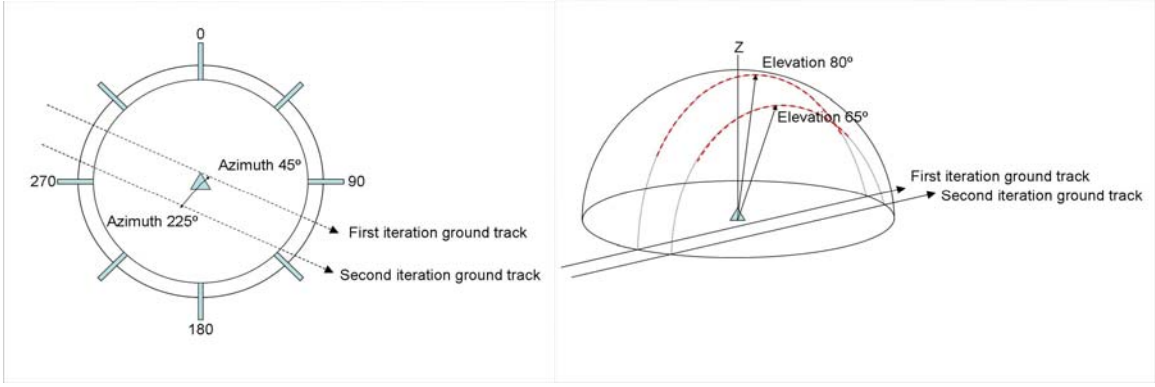

Figure 3 Angle residual errors

If the state correction calculated at the end of an iteration included a sufficient change in velocity, the propagated orbit was likely to cross an angle meridian shortly before or after the reference state did. During this interval, the angle residuals grow to the full modulo of the angle measurement. The result is an unusually large correction (compared to similar filter corrections using other data types) to the velocity components of the reference state. This correction leads to diverging angle residual behavior, ultimately causing the filter to fail to converge. This problem was overcome by evaluating data formatted as pseudo-state vectors.

Pseudo State Vector

Research into the possibility of alternative data representations to overcome persistent angle residuals discussed above led to Dr Wiesel's description of the pseudo-state position vector. If the data is in the form of Wiesel's pseudo-state: *Q* is variable, the pseudo-state position vector in the South-East-Z coordinate frame would be

$$\begin{aligned} \rho_S &= -\rho \cos\beta \cos\alpha \\ \rho_E &= \rho \cos\beta \sin\alpha \\ \rho_Z &= \rho \sin\beta \\ \vec{z}_{psuedo\ observation\ SEZ} &= \vec{\rho}_{SEZ} \end{aligned}$$

The inertial position vector of the target would then be

$$\vec{r}'_{IJK} = \vec{z}_{psuedo\ observation\ IJK} = R_{siteIJK} + [SEZtoIJK]\vec{\rho}_{SEZ}$$

The azimuth α, elevation β, and range ρ data are thus transformed into a pseudo-observation of the position *r*. The true error in $\vec{r}'_{IJK}$ is related to the true error in the actual data by a Jacobian matrix *J* given by

$$J = [SEZtoIJK]K$$

Where *K* is given by

$$K = \begin{pmatrix} -\cos\beta\cos\alpha & \rho\cos\beta\sin\alpha & \rho\sin\beta\cos\alpha \\ \cos\beta\sin\alpha & \rho\cos\beta\cos\alpha & -\rho\sin\beta\sin\alpha \\ \sin\beta & 0 & \rho\cos\beta \end{pmatrix}$$

The covariance for pseudo-position vector *r'* is then

$$Q' = JQJ^T \tag{3.28}$$

In this case, Q' from Equation (3.28) would replace *Q* in both Equations (3.16) and (3.25) above. And since the pseudo-state data is in the same form as the state vector *X*, the observation relation *G* becomes

$$G(\vec{X},t) = \left[I, \varnothing\right]\vec{X}$$

However, this transformation requires that range be a measured data element.

As Wiesel describes, it is possible to transform the data to closely resemble part of the state vector itself. The logical question here is why not use groups of three optical observations to establish a full pseudo-state vector using initial orbit determination techniques such as Vallado's Herrick-Gibbs algorithm or Gauss Angles-Only and use that full vector as our "data." The reason is that there is no obvious way to assign a statistically valid error variance to such a range "measurement" from a single telescope. Specifically, an instrument's error variances and biases must be ascertainable through calibration. For example, initial orbit determination calculations such as Gauss Angles Only or Double-R iteration are made with uncalibratable time intervals between observations that depend on any given target's mean motion and range. Differential correction techniques such as Least Squares or Bayes require data based on known instrument biases and error variances (sigmas) in order to compute the covariance matrix P so that a statistically valid confidence can be given to a future prediction. If a quantity that cannot be calibrated, then its statistical variance as a data element is unknowable, thus it cannot be included in a filter to establish an estimate with an associated statistical confidence. As the measured range ρ becomes large, the uncertainty in the line of sight direction remains constant, while constant values of σ_α and σ_β transform into ever larger

position uncertainties perpendicular to the line of sight. For large ρ, the covariance Q' describes a flat pancake normal to the line of sight vector.

Wiesel notes that a similar transformation cannot be done for data from an optical telescope, which typically includes only two angles such as right ascension and declination, or azimuth and elevation. Without range information, a transformation to recalculate r is not possible.

As described in an earlier section, the range residual vector containing angle information causes instability in the estimation filter. One possible solution to this problem is to re-arrange the data vector to remove angle information and replace it, preferably, with a data vector that resembles the state vector. Inconveniently, with a telescope employed in a typical manner, this is impossible. Thus a reasonable conclusion has been drawn by previous researchers such as Toso [10] that a space surveillance network consisting of typical telescopes providing data on LEO targets to an estimator is unfeasible.

However, optical instruments are being developed that can be employed in a manner that would allow continuous range measurement throughout an observation. Specifically, if a pair of slewing (tracking) telescopes were networked and operated to each observe the same target simultaneously from different locations, the intersecting line of sight vectors could be employed to directly measure range to the target vis-à-vis binocular observations.

Optical Ground Site Architecture

The characteristics of the Rocketdyne portable optical LEO tracker discussed in Chapter 2 and Appendix A make it possible to propose a unique tactical employment of these optical sensors in linked groups that provide range measurements during simultaneous observations with binocular viewing geometry. If a group of similar sensors that also processed data by commercial applications on portable computers were to share their tracking information over a network, then this group of sensors could effectively be combined into a single observer network that synthesized simultaneous observations into a single target track. Real-time network-based data sharing and collaboration is currently being done for example via open database connection (ODBC) applications.

The geometry of two optical trackers, sensor A and sensor B, linked to establish binocular observations of a target, S, is shown below in Figure 4:

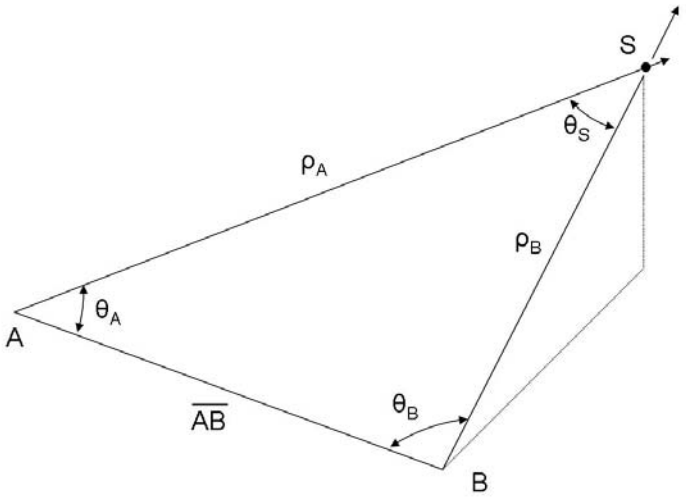

Figure 4 Paired sensor viewing geometry

The first consideration in ascertaining the range measurement errors for this observer site is the pointing accuracy of the LEO tracker, published as .00286 degrees angular position [9, pg 84]. However, these angle errors will have a varying effect on range data accuracy depending on the range itself and the distance between the two observing trackers.

The shape and size of the optical range "pixel" in the "along viewing axis" and "horizontal cross viewing axis" dimensions is shown in Figure 5 below. Given a 100km separation between sensor A and sensor B, the maximum along-viewing axis error is calculated to be 0.784km, and the maximum horizontal cross-viewing axis error is shown to be 0.035km. Thus the range variances form a football-shaped ellipsoid of error pointing at the sensors. This error has been approximated as a circle of error in the S-E plane (equal in each axis) with diameter equal to the magnitude of the along-viewing axis error, and error in the Z axis of a magnitude equal to the cross-viewing axis error. Azimuth and Elevation can be used to calculate line-of-sight vectors from sensor A and B because B's position is referenced back to A's position in the SEZ coordinate system.

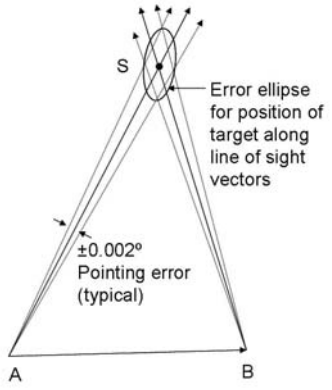

Figure 5 Observer network pointing error ellipse

Pointing errors in the Z axis create non-zero skew vector miss distances, as seen in Figure 6.

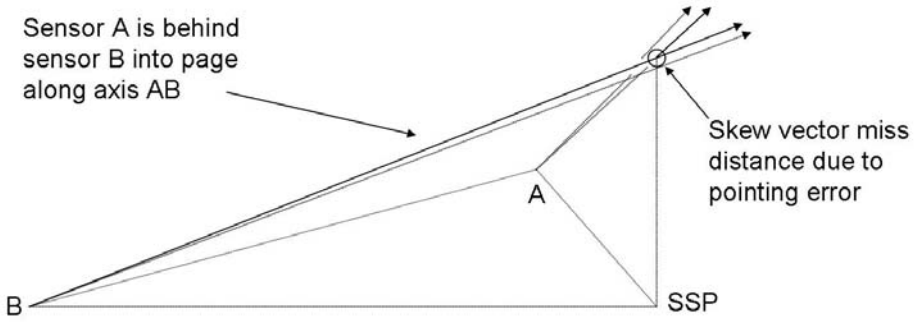

Figure 6 Skew vector miss distance

A displacement vector between two skew lines of sight and its associated magnitude can be calculated to give an expected error for this case. Per reference [11], a vector M is given by the cross product of the two known line of sight vectors L_A and L_B (see figure 6 below). Vector AB is known with magnitude 100km, and the length of vector AB along unit vector U_M is $\langle AB \cdot U_M \rangle$. Therefore, the displacement vector M is given by $\langle AB \cdot U_M \rangle U_M$.

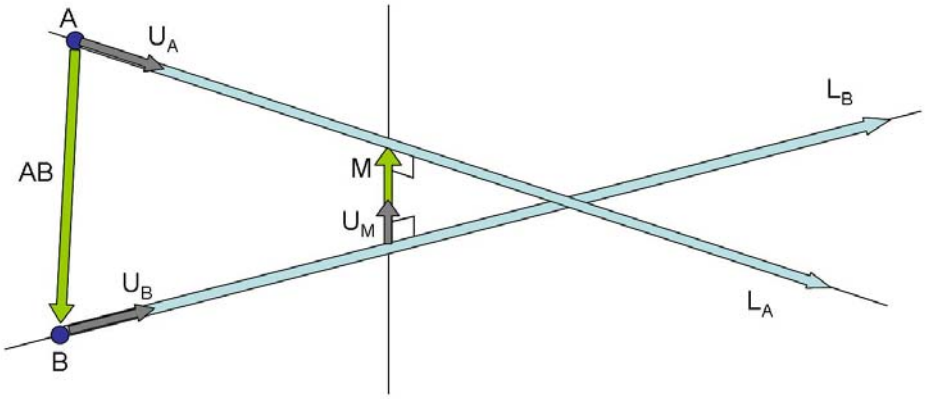

Figure 7 Skew vector geometry

This method of networking slewing optical trackers into binocular sensor pairs allows us to overcome the limitations of typical telescope observations to provide a calibratable range measurement for each observation of the LEO target. Note that it is important to ensure the line of sight vectors from each tracker will intersect despite pointing errors. If the target is observed along the line defined by the vector between the two trackers, pointing errors may cause the line of sight vectors to diverge on either side of the target. Thus as will be seen in the scenario development later, three sensors are deployed in a triangle at each observer site to guarantee a binocular viewing geometry.

Finite Differencing

Although observation data is formatted as a pseudo-state vector, large residuals returned by the filter algorithms may yet cause instability due to the reference orbit's sensitivity to large corrections, particularly in the velocity terms. Efforts to overcome filter instability due to large corrections led to Vallado's treatment of finite differencing [15, pg707]. Vallado's approach to orbit estimation includes a technique referred to as finite differencing wherein small changes are made to the state vector in order to eliminate sensitivity to large and small values within the state vector elements. For example, very small changes to an orbit's semi-major axis will have very large effects on the satellite's calculated motion due to the change in the mean motion.

A similar approach was applied to the state corrections calculated by Wiesel's algorithm. Observation processing of a LEO target using Wiesel's Non-Linear Least Squares algorithm rapidly diverged even with pseudo state vector data because the filter generated a large initial state correction (on the order of 10% of the position vector

magnitude) which with the LEO target's dynamics created a large and persistently expanding residual vector.

Instead, following Vallado's rationale, the state correction calculated by each algorithm was scaled down by an arbitrarily small fixed factor, *V*, (set between 1% and 10%) to keep the filter stable, incrementally marching an estimate toward convergence. As a result, Equation (3.17) from the Non-Linear Least Squares algorithm and Equation (3.27) from the Bayes algorithm were modified to include scale factor *V*. Thus for Non-Linear Least Squares:

$$\delta\vec{X} = ((T^T Q^{-1} T)^{-1} T^T Q^{-1} \vec{\mathrm{r}})V = PT^T Q^{-1} \vec{\mathrm{r}} \cdot V \tag{3.29}$$

And for Bayes:

$$\delta\vec{X}(t_o) = P(+)(P^{-1}(-)(\vec{X}(-) - \vec{X}_{ref}) + \sum_{i=1}^{N} T_i^T Q_i^{-1} r_{zi}) \cdot V \tag{3.30}$$

By using this technique, interim state corrections calculated by Wiesel's algorithm provided the *proportion* of each state element's correction, while the scale factor *V* determined the *magnitude* of the correction. This technique allowed the filters to be tuned for stability.

Convergence Criteria

The convergence criteria for both the batch and sequential algorithms was the reduction of the state correction for each element in the state vector, $\delta\overline{X}_i$, to below a percentage of its associated variance in the calculated covariance matrix. In use, the estimate was deemed to have converged if the correction for each state element was less than 10% of its associated variance. However, the convergence criteria were tightened to

1% and 0.5% in experiments to ascertain how much the quality of the estimate improved. Filter performance and stability were adjusted by varying the $\delta\vec{X}$ scale factor *V* as well as the convergence criteria.

Scenario

The simulation scenario was built on the premise of tracking a newly-launched LEO target for which no historical ephemeris existed. The LEO target observed in this scenario was the Space Shuttle mission STS-109 in a 580km orbit with an inclination of 28.5deg and an eccentricity of 0.00733. A representative ground track was shown above in Figure 1 on page 25.

The objective was to track the target using one space tracking radar and a network of optical observers. Based on the research of Foster [3], the radar matched the performance of the Russian Don-2N space vehicle tracking radar, with full hemispherical coverage, detection range of 600 – 1,000 km for a 5-cm space object, and accuracies of 0.02º-0.04º angular position and 200 meters in range.

The optical ground sites in the simulation are equipped with an observer network of Rocketdyne's prototype 58-lb portable LEO satellite tracker, with adjustable slew rates to six degrees per second, implying the ability to track orbiting targets at all altitudes. Track errors for each sensor are less than 50 μrad (0.00286 deg) peak to valley for the duration of a target pass. A scale representation of a typical ground site with triangular LEO tracker deployment, 100km on a side, is shown in Figure 7. The simulated position measurement error for the LEO tracker is 1 km in all three axes.

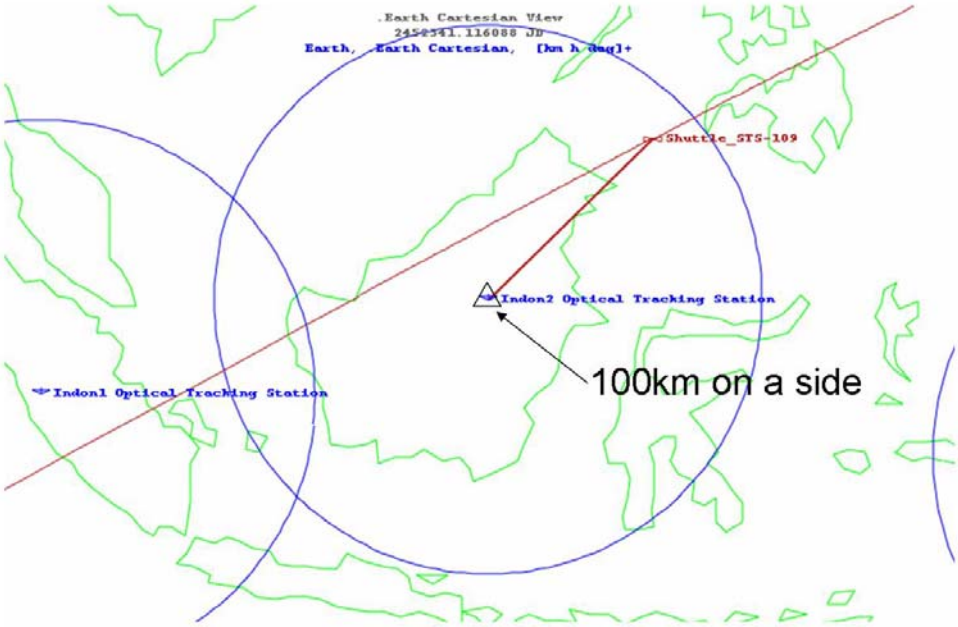

Figure 8 Scale view of Indonesia 3 ground site

The simulation targeted Space Shuttle mission STS-109 based on archived NORAD 2 line element sets, which allowed the complete mission, including all maneuvers, to be loaded into SOAP to evaluate filter estimates against actual target position using a separate propagator and application.

Covariances for the radar and optical sites as used in the MATLAB code are shown below in Table 2.

Table 2 Covariances for radar and optical sites used in MATLAB code

	Don 2 N Radar	Optical Sites	
		Each portable tracker	Networked sensors
Range accuracy (σ)	200 m	NA	784 m
Angle accuracy (σ)	0.02 deg	0.00286 deg (50 μrad)	NA

Code Validation

Although the Non-Linear Least Squares and Bayes filter codes were not rigorously verified (such as Monte Carlo simulations to ascertain inherent noise and bias in the filters), a sanity check on the performance of the propagator was accomplished by comparing the performance of the 'ground_rhs.m' routine in MATLAB (ODE45) dynamics model to that of SOAP output after propagating the orbit forward for 57 minutes (between the China observation site and the Brazil 1 observation site) from the same initial conditions. The results are shown below in Table 3. The variation is less than 1% for each element of the state vector.

Table 3 Variation of Target States propagated by SOAP vs 'ground_rhs.m' (MATLAB ODE45)

	x (km)	y (km)	z (km)	$\dot{x}$ (km/s)	$\dot{y}$ (km/s)	$\dot{z}$ (km/s)
Initial State for both propagators	-896.353	-6214.373	2896.839	7.1541768	-1.8862072	-1.6998204
Target State after 57 minutes calculated by SOAP SGP-4 propagator	-2646.429	6140.948	-1635.907	-6.6251469	-2.0649525	3.1600054
Target State after 57 minutes calculated by MATLAB ODE 45 propagator	-2666.283	6134.723	-1626.422	-6.61541	-2.08744	3.16598
Variation	19.854	6.225	9.485	0.009737	0.022488	0.005975

The variation between the 'ground_rhs.m' propagator and SOAP's SGP-4 propagator contribute to the errors seen in evaluation of the Shuttle target scenario explored herein. Given that the code is not verified, the results provide by the code from

evaluation of the Shuttle target scenario does not reflect a statistically valid representation of either filters performance.

Summary

Batch and sequential data filters were coded into MATLAB to process orbit observations of a LEO target to evaluate the feasibility of orbit estimation and rendezvous mission planning based on observations by a space surveillance network of optical sensors. The filter algorithms were modified to enhance stability by processing pseudo-state vector data and making only small corrections to each interim state estimate based on the theories of Wiesel and Vallado, although the filter code has not been statistically validated by, for example, a Monte-Carlo series of simulations.

The filters were evaluated against a realistic albeit hypothetical LEO target represented by Space Shuttle mission STS-109, modeled in the SOAP software application to check calculated estimates against actual position data.

IV. Analysis and Results

Chapter Overview

Although LEO satellite orbit estimation using non-linear least squares and sequential estimation techniques is well addressed in the literature, the performance of these filters depends on several factors including the fidelity of the propagator and the accuracy as well as the quantity of data. This research focused on a scenario that featured a realistic target (archived Shuttle mission ephemeris), tracked by realistic albeit hypothetical ground sites. Realistic targeting mission planning must include the possibility that the target maneuvers. Simulation results feature the discovery of large errors between the true state and the estimated state between ground sites likely due to differences between the propagators used in the simulation, highlighting the need to include higher-fidelity variations in the 'ground_rhs.m' routine.

Performance of Non-Linear Least Square Algorithm

Because the Bayes sequential algorithm required a prior estimate and associated covariance matrix to process new data, observations from the radar ground site were provided to the Non-Linear Least Squares filter to process as a batch. Initial batch filter performance with 3.5 minutes of radar observations spaced 1 second apart, and scale factor V = 1 as defined in Equation (3.29) was poor. Vallado [16] predicted that although an estimate could be ascertained from dense short arcs of data, he concluded that a minimum of 6 minutes of data were necessary to converge on an accurate orbit estimate. Thus the filter's difficulty achieving a more accurate estimate could be due to the lack of

a large enough data arc. Furthermore, both Wiesel [18, pg 68] and Vallado [15, pg 710] state that "convergence is not guaranteed" given any group observations of a target.

Although the Wiesel algorithm appears to approach convergence, it fails when the variable instrument covariance matrix Q' approaches singularity (the estimated state puts the LEO target's position below the observer site's horizon). Note trends in the batch algorithm performance (figures below). The Bayes is almost identical except in its use of a previous P matrix, and has not been evaluated here because the output estimate and covariance from the Least Squares algorithm is still too poor to properly use in Bayes.

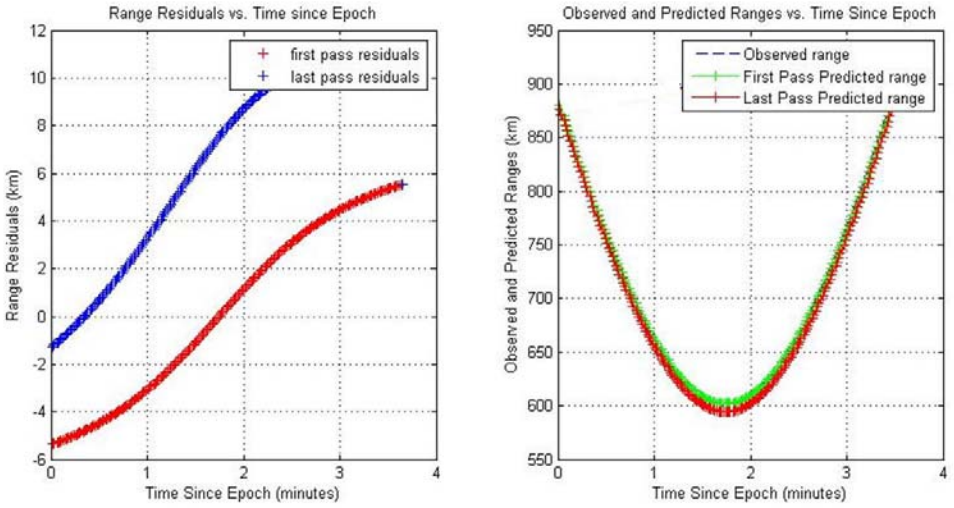

Figure 9 Second iteration residuals and fit in the batch filter

The filter residuals vary by 10 km, but are being fit to a quadratic curve representing the target's range as it flies over the observer location. In these results, in the right-side figures, the "first pass predicted range" plots directly over the "observed range" plot. After each iteration through the data by the filter, an adjustment is made to the state at epoch, effectively changing the starting location for the target as it enters the observer's sensor range, causing the quadratic curve representing the calculated

(propagated) to shift. As can be seen in the next series of figures, the filter diverges rapidly as the calculated range residuals between the curves representing the observed position data and the calculated position data grow with each iteration's correction to the state.

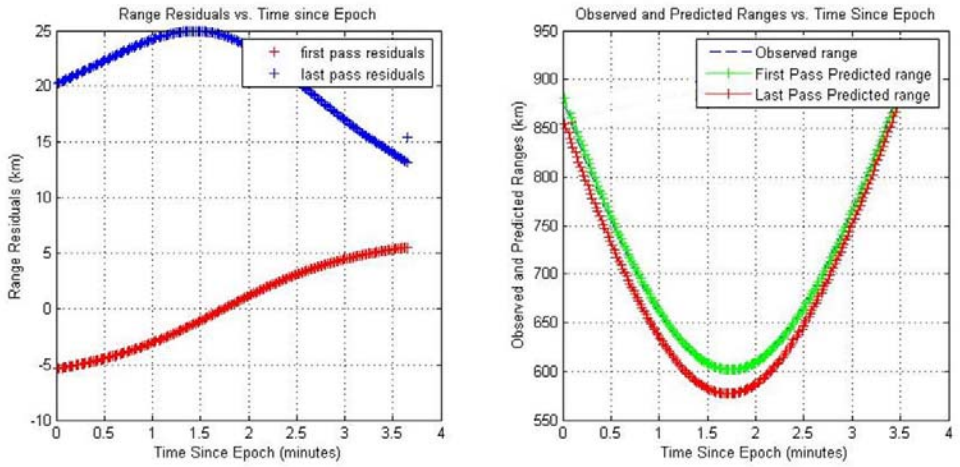

Figure 10 Fourth iteration residuals and fit in batch filter with V=1

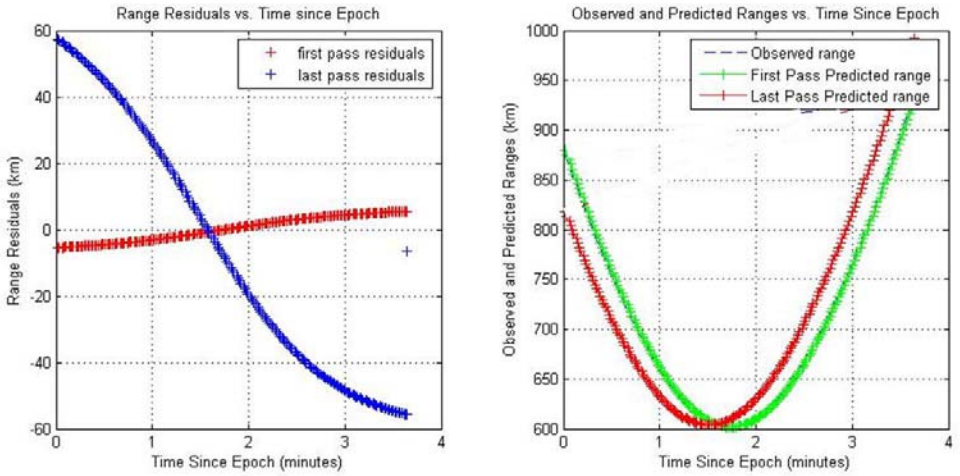

Figure 11 Sixth iteration residuals and fit in the batch filter with V =1

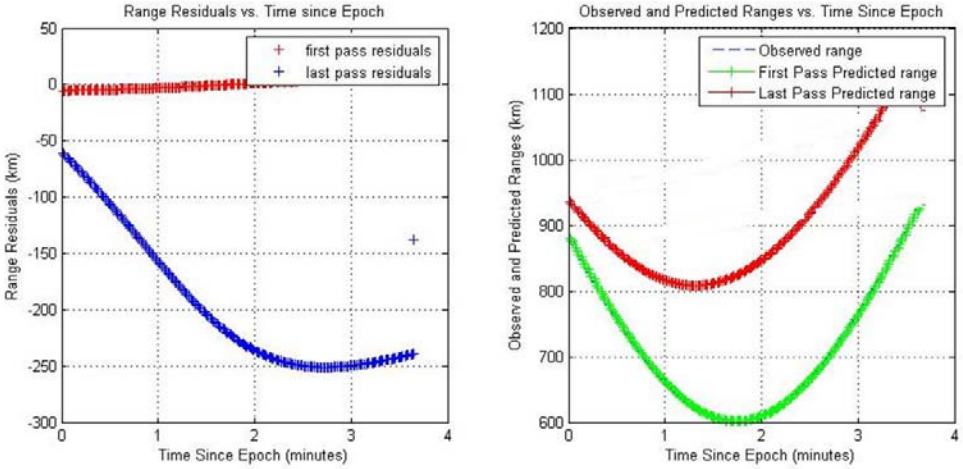

Figure 12 Eighth iteration residuals and fit in the batch filter with *V*=1

After 8 iterations, the inverse of the covariance matrix P approached singularity, Interestingly, the correction to the reference state, $\delta\vec{X}$, after each iteration looks like an average of the residuals for that iteration (as opposed to seeing $\delta\vec{X}$ diminish as in Vallado's differential correction routine). Whereas Wiesel's algorithm calculates $\delta\vec{X}$ strictly from known dynamics and the observation relationships to the State, Vallado uses finite differencing, which makes an arbitrarily small correction to the reference state for each iteration to "march" toward convergence. Vallado's technique is shown to be effective in his text, however Wiesel's approach, employed in this project, while more difficult to employ, is more statistically stringent. Following these initial results, scale factor V was set to 0.1, implying that only 10% of the change to the reference state calculated by the Wiesel non-linear least squares algorithm would be made for the next iteration.

After 4 iterations, data residuals were small, as expected, but the state corrections were not moving the estimated state toward a position that immediately minimized the residuals (see Figures 12-15 below).

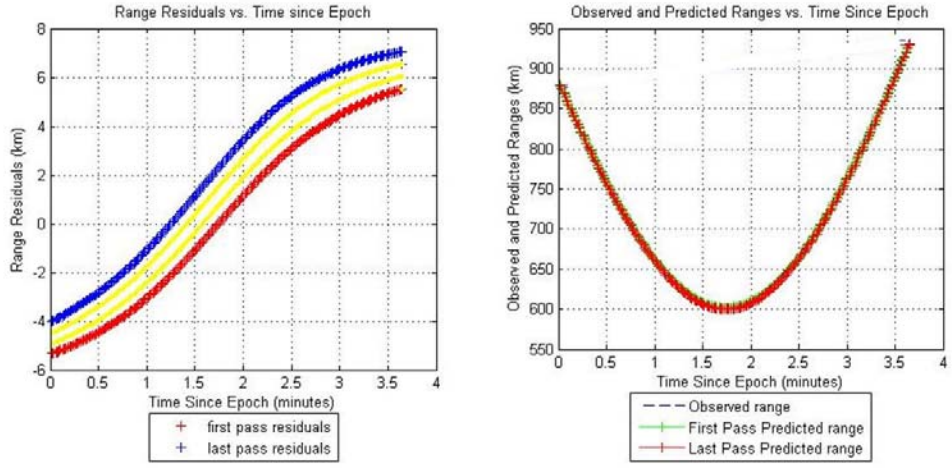

Figure 13 Fourth iteration residuals and fit in batch filter with *V*=0.1

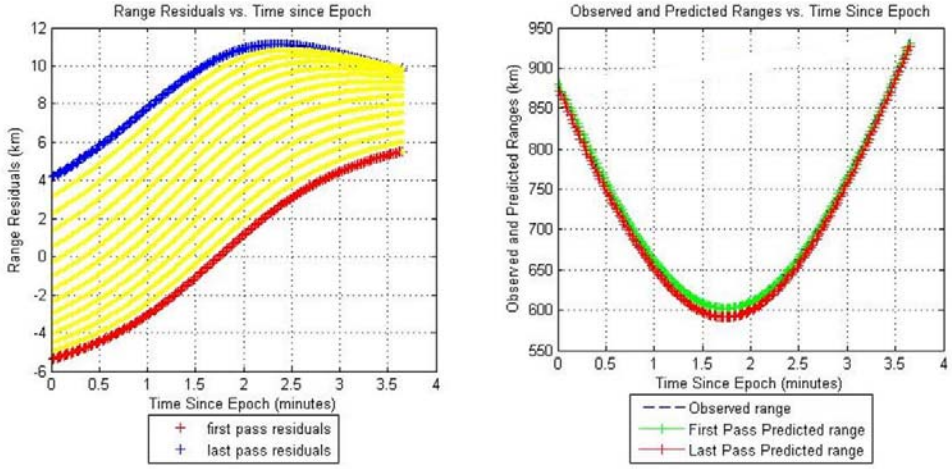

Figure 14 15th iteration residuals and fit in batch filter with *V*=0.1

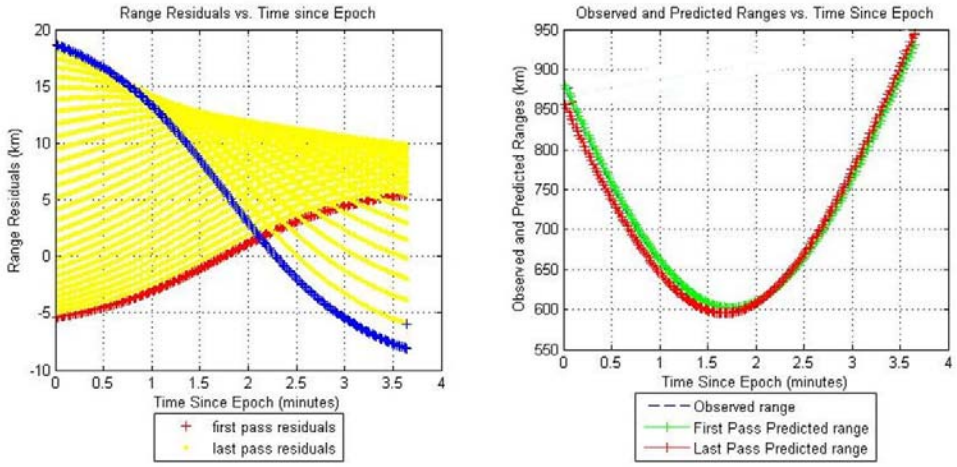

Figure 15 30th iteration residuals and fit in batch filter with *V*=0.1

With *V*=0.1, the filter's interim state corrections behaved as they had with V=1, however, the variance of the residuals was 20km instead of 130km. As seen, however, the quadratic curve fit of the propagated state versus the observed state across the arc of data still created a persistent residual vector that was exacerbated by each correction. Because the convergence criteria required corrections be less than 10% of the associated covariance for each element of the state vector, the fit of the estimate converged on by the filter to the observed data was poor (Figure 15).

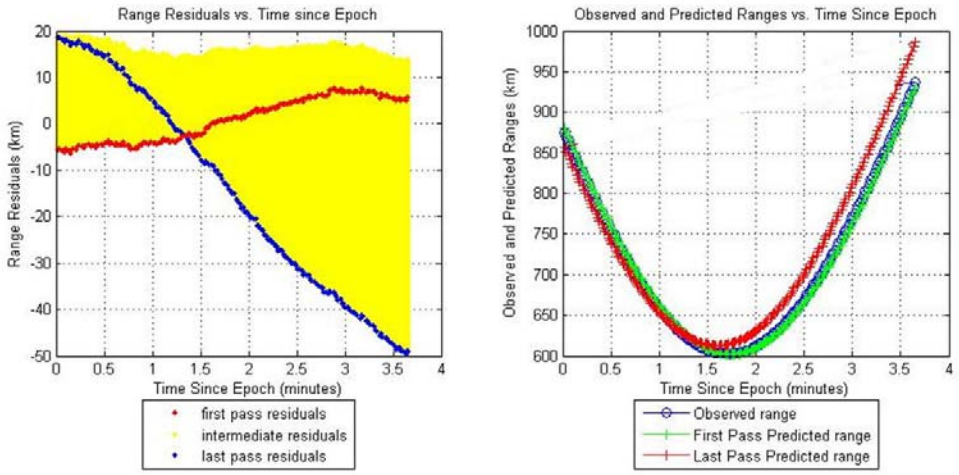

Figure 16 Converged estimate residuals and fit (noisy observation data)

Next, a change in the reference coordinate system from the SEZ frame to the IJK frame was explored for its effect on the filter's ability to converge on an accurate estimate. Processing noisy observation data (subject to 1km errors in each axis), with convergence criteria set to 10% of each state element's covariance and scale factor V=0.1, the filter adjusted the estimate to normalize the residuals around a zero mean, although the refined estimate did not reduce the variance of the residuals, which remained at 3 km.

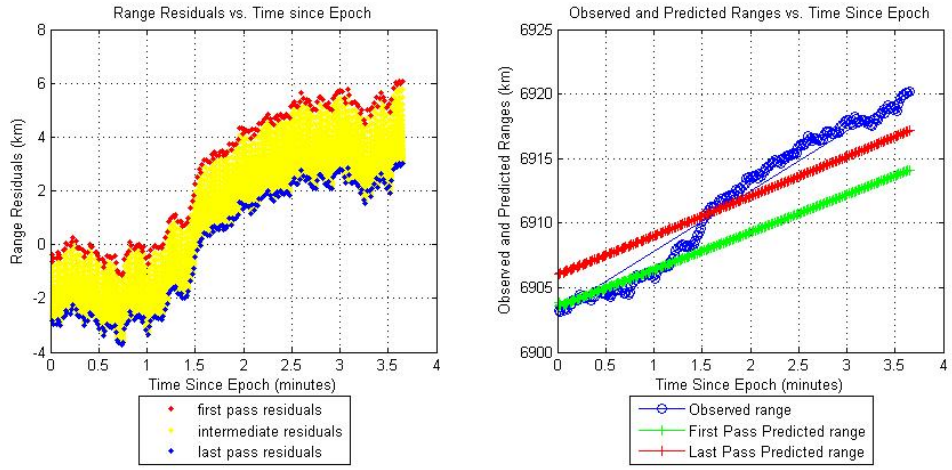

Figure 17 Converged results for data in IJK frame

Processing Scenario Observation Data with Non-Linear Least Squares and Bayes

The initial orbit state propagated by the 'ground_rhs.m' routine in MATLAB for this test of the code's performance against the Shuttle target is (show state vector) and is depicted below in Table 4.

Table 4 Initial State for test of code

	x (km)	y (km)	z (km)	$\dot{x}$ (km/s)	$\dot{y}$ (km/s)	$\dot{z}$ (km/s)
Initial State	-896.353	-6214.373	2896.839	7.1541768	-1.8862072	-1.6998204

The Bayes filter was evaluated by first establishing an orbit estimate from radar observations made from the China site, and then propagating that estimate and covariance to an epoch corresponding to the next observation of the target, over the Brazil 1 observation site 54 minutes later. Observation data from the Brazil 1 site was processed both sequentially and as a batch using Non-Linear Least Squares to compare the estimates from both filters. Similarly, the estimate established by the Brazil 1 observations was brought forward to the next target observation by the Brazil 2 site, 6 minutes later. The filter converged in each case with a 3.5 minute arc of observation data in the IJK frame from each site, spaced 1 second apart. Scale factor was set to V=0.1, and the convergence criteria was set to 1%.

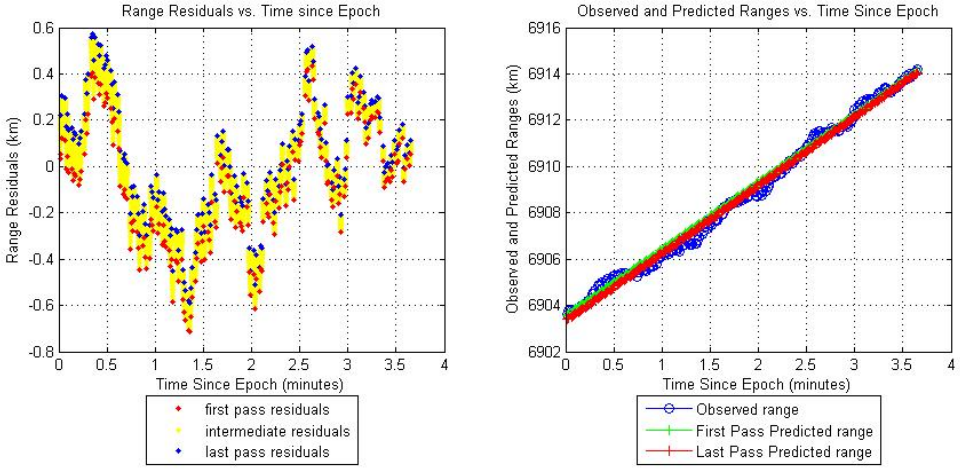

Figure 18 Residuals from LSQ filter for China site radar observations

Data processed from the optical observer sites Brazil 1 and Brazil 2 by the Non-Linear Least Squares filter are shown below in Figures 19 and 20.

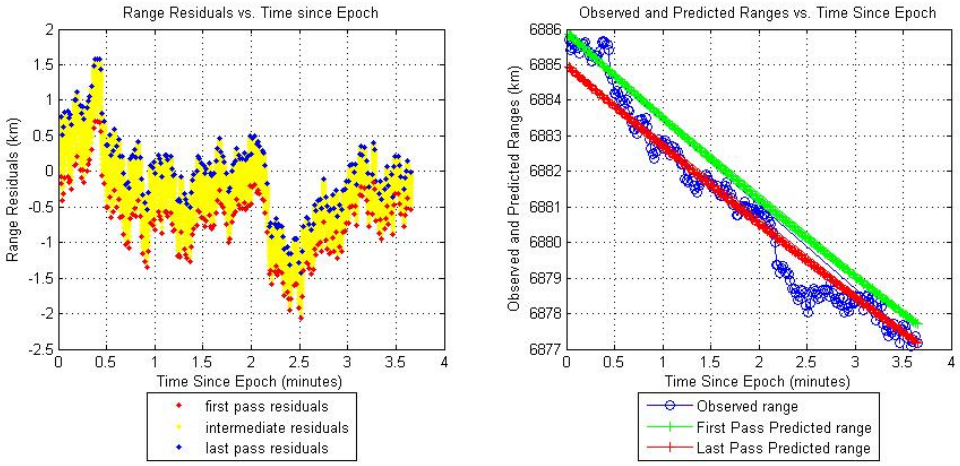

Figure 19 Residuals from LSQ filter for Brazil 1 site observations

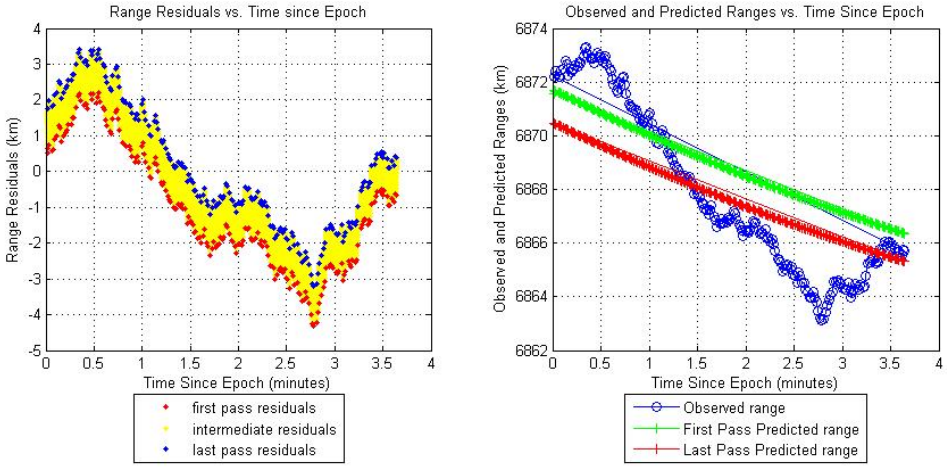

Figure 20 Residuals from LSQ filter for Brazil 2 site observations

Comparatively, the Least Squares estimate from the China site data and the data from the Brazil 1 observer site (collected at an epoch 54 minutes later) were processed by the sequential Bayes filter, with the results shown in Figure 21 below.

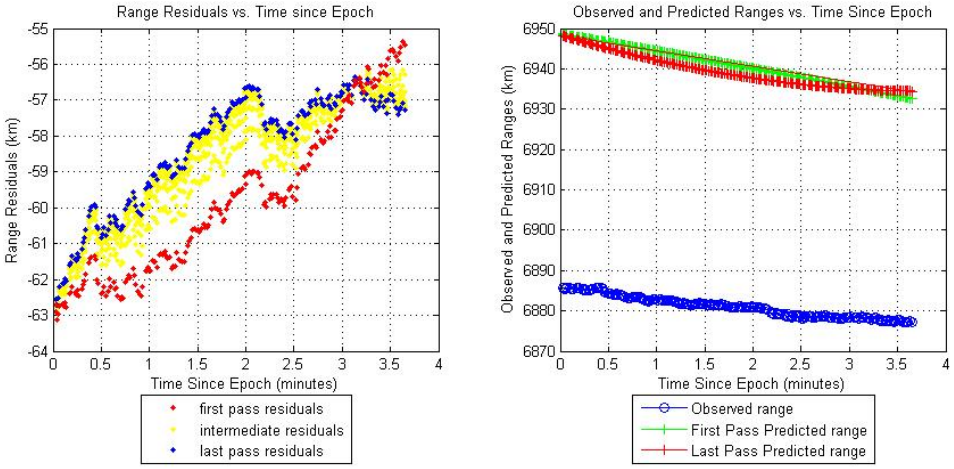

Figure 21 Residuals from the Bayes filter for Brazil 1 site observations

Similarly, the Least Squares estimate from the Brazil 1 site were processed with new data from the Brazil 2 site (collected at an epoch 6 minutes after that of the Brazil 1 site) by the Bayes filter with the results shown in Figure 22.

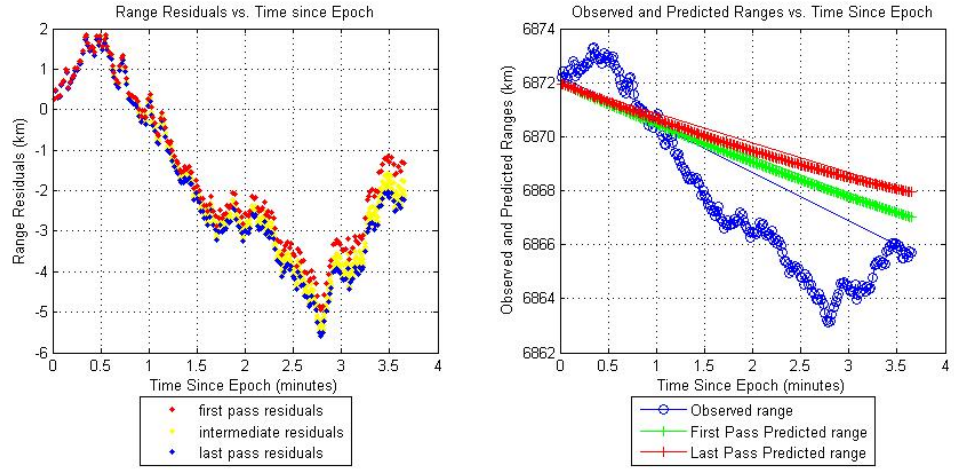

Figure 22 Residuals from the Bayes filter for Brazil 2 site observations

Comparison of estimates propagated forward three hours

Evaluation of data starting at an epoch corresponding to observations of the target by the Brazil 1 site provide insight to the estimates established by the Non-Linear Least Squares estimate as well as the Bayes filter, shown below in Figure 23. The baseline orbit for comparison is the initially determined orbit state at the epoch of the target's pass over the Brazil 1 observation site a propagated by the MATLAB 'ground_rhs.m' ODE45 propagator, with estimated orbits shown in terms of range error from the baseline orbit.

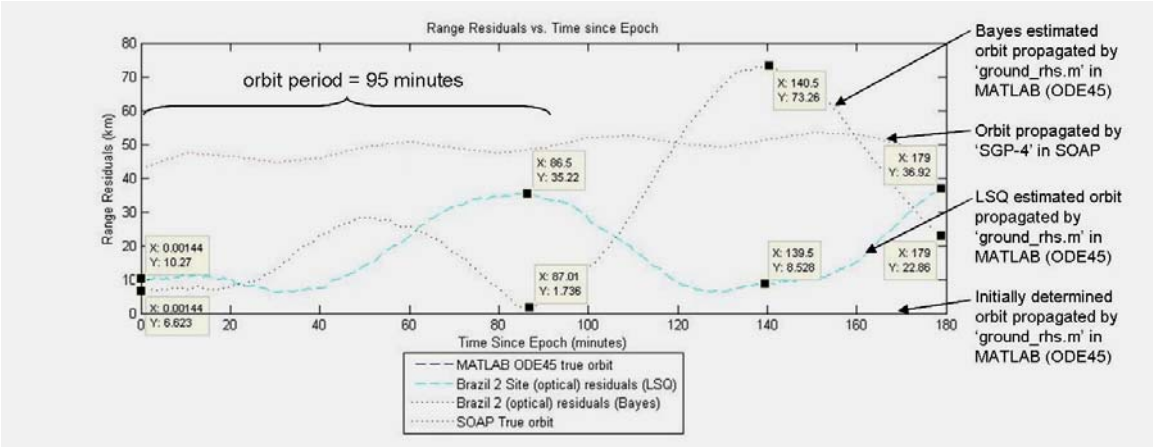

Figure 23 LSQ and Bayes estimates propagated forward 3 hours from Brazil 1 site observations epoch

The true orbit is also shown as represented in SOAP at the epoch of the target's observation by the Brazil 1 site, propagated forward by SGP-4. The orbit estimates as determined by both the Non-Linear Least Squares filter and the Bayes filter differed from the orbit propagated by the 'ground_rhs.m' routine's dynamics by 7 and 6 kilometers respectively after one orbit, which exceed the expected range of a micro-satellite's terminal sensor. However, these initial differences between the true state and the

estimated state, particularly even the small differences in the estimated target velocity state, cause enormous variation in the projected position of the target within the three hour propagation timeline. Furthermore, the "true" target state represented in SOAP was approximately 45km off from the target location calculated in the MATLAB propagator.

The covariances associated with the estimate established by observations over the Brazil 1 site and propagated forward to the next set of observations are below. As expected, the covariances for position and velocity grow as it is propagated forward in time, and is then reduced following the estimate established by observations over the Brazil 2 site.

Covariance matrix *P* calculated by Nonlinear Least Squares Estimate algorithm for observations of the target over the Brazil 1 site:

$$P = \begin{bmatrix} 5.48\text{E-}03 & -3.37\text{E-}05 & 9.41\text{E-}06 & -4.39\text{E-}06 & 5.00\text{E-}07 & -1.46\text{E-}07 \\ -3.37\text{E-}05 & 5.56\text{E-}03 & -2.72\text{E-}05 & 4.99\text{E-}07 & -5.84\text{E-}06 & 4.75\text{E-}07 \\ 9.41\text{E-}06 & -2.72\text{E-}05 & 5.48\text{E-}03 & -1.46\text{E-}07 & 4.75\text{E-}07 & -4.37\text{E-}06 \\ -4.39\text{E-}06 & 4.99\text{E-}07 & -1.46\text{E-}07 & 4.84\text{E-}08 & -3.09\text{E-}10 & 9.99\text{E-}11 \\ 5.00\text{E-}07 & -5.84\text{E-}06 & 4.75\text{E-}07 & -3.09\text{E-}10 & 4.95\text{E-}08 & -3.88\text{E-}10 \\ -1.46\text{E-}07 & 4.75\text{E-}07 & -4.37\text{E-}06 & 9.99\text{E-}11 & -3.88\text{E-}10 & 4.84\text{E-}08 \end{bmatrix}$$

P(-) covariance matrix at the start of Bayes estimate algorithm for observations of the target over the Brazil 2 site, after being propagated forward 6 minutes:

$$P(-) = \begin{bmatrix} 40676 & 41762 & 42846 & 43929 & 45009 & 46087 \\ 41762 & 42877 & 43990 & 45101 & 46211 & 47318 \\ 42846 & 43990 & 45132 & 46272 & 47411 & 48547 \\ 43929 & 45101 & 46272 & 47441 & 48608 & 49773 \\ 45009 & 46211 & 47411 & 48608 & 49804 & 50997 \\ 46087 & 47318 & 48547 & 49773 & 50997 & 52219 \end{bmatrix}$$

Covariance matrix *P* calculated by Bayes algorithm for observations of the target over the Brazil 2 site:

$$P = \begin{bmatrix} 3.71E\text{-}07 & 2.82E\text{-}07 & 1.93E\text{-}07 & 1.04E\text{-}07 & 1.48E\text{-}08 & -7.44E\text{-}08 \\ 2.82E\text{-}07 & 2.15E\text{-}07 & 1.49E\text{-}07 & 8.17E\text{-}08 & 1.48E\text{-}08 & -5.20E\text{-}08 \\ 1.93E\text{-}07 & 1.49E\text{-}07 & 1.04E\text{-}07 & 5.95E\text{-}08 & 1.49E\text{-}08 & -2.97E\text{-}08 \\ 1.04E\text{-}07 & 8.17E\text{-}08 & 5.95E\text{-}08 & 3.72E\text{-}08 & 1.49E\text{-}08 & -7.32E\text{-}09 \\ 1.48E\text{-}08 & 1.48E\text{-}08 & 1.49E\text{-}08 & 1.49E\text{-}08 & 1.50E\text{-}08 & 1.50E\text{-}08 \\ -7.44E\text{-}08 & -5.20E\text{-}08 & -2.97E\text{-}08 & -7.32E\text{-}09 & 1.50E\text{-}08 & 3.74E\text{-}08 \end{bmatrix}$$

Analysis

The Δv's associated with the variances reflected in Figure 23 in are shown in Table 5 below (for 2-impulse Hohmann transfer, assuming in-plane maneuvers). According to Doug Brown of the DART Launch Systems group, the 700lb DART rendezvous vehicle has a 500 m/s Delta-V budget for rendezvous and a 50m/s Delta-V budget for proximity operations.

Table 5 Delta V's associated with propagated estimate errors

	Total Delta V (m/s)
Propagated Least Squares Estimate	
86 minutes	18.9
140 minutes	4.7
180 minutes	21.1
Propagated Bayes Estimate	
86 minutes	0.87
140 minutes	38.8
180 minutes	12.1

The limiting factor for the terminal phase of a microsatellite rendezvous in the scenario examined by this study is likely the terminal sensor, versus the Delta-V budget, if the error in the estimate can be reduced for a second transfer maneuver.

The state of a LEO target can be ascertained by a surveillance network of optical sensors tactically deployed for simultaneous (binocular) observations. The resulting estimated state can be known fairly accurately after each over flight, whether by a Least Squares filter or a sequential Bayes filter to serve as a reference for a rendezvousing satellite. The results of this study show that although the Bayes filter provides an estimate closer to the true state at epoch than that calculated by the Least Squares filter, variations in the position and velocity states cause wide variations from the true state as each estimate is propagated forward.

V. Conclusions and Recommendations

Conclusions

The key to the targeting problem is knowing the state well enough to execute a rendezvous maneuver to within the interceptor's terminal sensors. AFIT research, including, for example, that of Foster has attempted to address the problems inherent in targeting a non-cooperative target. Foster approached the state estimate as a non-linear least squares problem. The drawback with non-linear least squares is twofold: 1) timeliness: since more data points lead to higher accuracies, the tendency is to take the time to gather a much data as possible; and 2) stability, because least squares estimation is done as a batch process, the accuracy of the final estimate of a dynamic system begins to degrade as soon as its is determined per Equation (3.19):

$$P((-),t_o) = \Phi(t_o, t_{o-})P(t_{o-})\Phi^T(t_o, t_{o-})$$

In the case of LEO satellite state estimation, an estimate's covariance will grow such that the position estimate may vary by tens of kilometers within one orbit.

This research extended an Air Force Institute of Technology Department of Aeronautics and Astronautics (AFIT/ENY) effort begun in 2003 to model space tracking system architectures in the MATLAB programming language to answer two key questions. First, can a non-cooperative target's state be estimated well enough from data gathered by a network of optical sensor sites to rendezvous a microsatellite interceptor close enough for its terminal sensors to find and track the target to a merge? Currently, filter and propagator performance are less than ideal. The model's filters and dynamics do not provide an estimate of the target's state that would get an intercepting

microsatellite to within a nominal 1km range. Future work on this model focusing on a refined dynamics propagator as well as a less conservative state correction scale factor (V) may improve the miss distance.

The second question focused on fuel limits, specifically; can the microsatellite be guided to the merge within its delta-v budget? Although ancillary to the ground-based orbit determination process explored in this research, calculations of the delta-v needed for a second in-plane 2-impulse Hohmann transfer to make up the miss distance would be within, for example, the DART experiment microsatellite vehicle's 50 m/s proximity operations delta-v budget, but would thus likely exhaust the microsatellite's maneuver capability. However, given one radar site and a set of optical sensors (such as Boeing-Rocketdyne's 58-lb prototype portable optical satellite tracker [9]), LEO target tracking for counter-space rendezvous mission planning is probably feasible given a more highly refined filter.

Recommendations for Future Research

Future research should focus on the efficacy of available commercial propagators and estimators, such as the RTOD application discussed in Chapter 2, with an emphasis in combing them with a network of sensors working together over a network via a sequential database application or similar technique to simultaneously share and process observation data.

With regard to this implementation of Non-Linear Least Squares and Bayes filters in MATLAB, the code should be validated using a Monte-Carlo series of runs, followed

by exploration of how the accuracy of the sequential filter's estimates vary with the number of observer sites and the number of observations per observer site.

Summary

This research extended an Air Force Institute of Technology Department of Aeronautics and Astronautics (AFIT/ENY) effort begun in 2003 to model space tracking system architectures in the MATLAB programming language to investigate the minimum requirements to establish a satellite tracking system architecture for a microsatellite to rendezvous with a non-cooperative target satellite. A prototype optical tracking system was reviewed with emphasis on a proposed tactical employment that could be used by technologically unsophisticated state or non-state adversaries. With the tracking system architecture selected, simulated tracking data was processed with a Non-Linear Least Squares batch orbit estimation algorithm and a Bayes sequential orbit determination filter to update the target satellite's state vector. Currently, the model's filters and dynamics do not provide an estimate of the target's state that would get an intercepting microsatellite to within a nominal 1km range. However, the concepts developed and implemented here could be applied by a more sophisticated space user, equipped with commercially available highly accurate propagators and estimators to achieve a realistic counterspace surveillance and rendezvous planning capability. Future studies should evaluate the implementation of these techniques with commercially available applications to assess their efficacy as space rendezvous mission planning tools.

Appendix A: T-Mount Current Specification from Reference [8]

T-Mount Current Specifications
Boeing/Rocketdyne
R.Tansey
(*data here reproduced from reference* [8, pg 84])

1. Pointing accuracy (Assuming lat, long, and sidereal time are correct)

	Measured Peak to Valley Radius	Estimated RMS Radius
Pointing anywhere in the sky after sync to stars	300 µrad	NA
Orbit track after single star sync	35 µrad	12 µrad
Orbit track after 10 sec duration	25 µrad	8 µrad

2. Software

2.1	Track of satellite is followed regardless of illumination/magnitude
2.2	Astronomy software with Hubble star catalog. Used for sync to stars (Windows 95/NT or 3.1)
2.3	Internal mount controller software. Astronomy control language protocol updatable via pcmcia. Used for feedback control with 22 bit encoders to maintain accurate position, velocity, and acceleration.
2.4	Satellite Tracking software Six element sets from Cheyenne are used to generate orbit. Mount follows orbit based on update from calculated values Horizon to horizon pointing 6 deg/sec max slew
2.5	Video Tracker Automatic update, commands drives [*sic*] mount to center of display Independent of orbit calculations Requires solar or “other” visible return from satellite Useful for maneuvering satellites. We were able to follow Mir after a 79 sec orbit burn
2.6	Other software features “quick look” to visually display ground track, and az, el “quick sync” to sync to star near rise point of satellite, allows fast accurate acquisition of orbit

3. Sensitivity

Current capability with .01 lux nonintensified ccd on 8” tele	mag 5/6
Intensified ccd on 12” tele (planned future work)	mag 7/8
possible	mag 9/10

4. Equipment used for current system

Pentium/120	Three b/w video monitors	Mount (volume 3 ft x 3 ft x 2 ft
sound card	joystick	8” telescope with ccd
video card	modem	2” lens with intensified ccd
computer monitor		

5. Operators needed: ……..one

Appendix B: Equations of Variation

In Jacobian [A], the elements of A, A(i,j) are the partial derivatives of the ith equation of motion with respect to the jth element of the state vector.

$$\frac{d}{dt}\vec{X} = f_{2-Body}(X) = \begin{pmatrix} \vec{v} \\ \dfrac{-\mu\vec{r}}{r^3} \end{pmatrix}_{6\times 1} \tag{B.1}$$

$$\frac{d}{dt}\Phi(t,t_o) = \dot{\Phi} = A(t)\Phi(t,t_o) \tag{B.2}$$

Per Wiesel page 86, $A(t) = \begin{pmatrix} \varnothing & I \\ A_{rr} & \varnothing \end{pmatrix}_{6\times 6}$ (B.3)

$$A_{2-Body} = \nabla f_{2-Body} = \frac{\partial f}{\partial x}\vec{i} + \frac{\partial f}{\partial y}\vec{j} + \frac{\partial f}{\partial z}\vec{k} + \frac{\partial f}{\partial \dot{x}}\vec{i} + \frac{\partial f}{\partial \dot{y}}\vec{j} + \frac{\partial f}{\partial \dot{z}}\vec{k} \tag{B.4}$$

$$\left[A(t)\right]_{6\times 6} = \begin{bmatrix} \dfrac{\partial}{\partial x}I \\ \dfrac{\partial}{\partial y}J \\ \dfrac{\partial}{\partial z}K \\ \dfrac{\partial}{\partial \dot{x}}I \\ \dfrac{\partial}{\partial \dot{y}}J \\ \dfrac{\partial}{\partial \dot{z}}K \end{bmatrix} \begin{bmatrix} v_I & v_J & v_K & \dfrac{-\mu r_I}{r^3} & \dfrac{-\mu r_J}{r^3} & \dfrac{-\mu r_K}{r^3} \end{bmatrix} \tag{B.5}$$

$$A_{rr} = \begin{pmatrix} \dfrac{-\mu}{r^3} + \dfrac{3\mu r_I^2}{r^5} & \dfrac{3\mu r_I r_J}{r^5} & \dfrac{3\mu r_I r_K}{r^5} \\ \dfrac{3\mu r_I r_J}{r^5} & \dfrac{-\mu}{r^3} + \dfrac{3\mu r_J^2}{r^5} & \dfrac{3\mu r_J r_K}{r^5} \\ \dfrac{3\mu r_I r_K}{r^5} & \dfrac{3\mu r_J r_K}{r^5} & \dfrac{-\mu}{r^3} + \dfrac{3\mu r_K^2}{r^5} \end{pmatrix} \tag{B.6}$$

Equations of variation for J2

Recall that $f_{J2}(X) = a_{J2} = \begin{bmatrix} \frac{-3J_2\mu R_\oplus^2 r_I}{2r^5}\left(1-\frac{5r_K^2}{r^2}\right) \\ \frac{-3J_2\mu R_\oplus^2 r_J}{2r^5}\left(1-\frac{5r_K^2}{r^2}\right) \\ \frac{-3J_2\mu R_\oplus^2 r_K}{2r^5}\left(3-\frac{5r_K^2}{r^2}\right) \end{bmatrix}$ (B.7)

$$A_{rr\,J2} = \nabla f_{J2} = \begin{pmatrix} \varnothing & \varnothing \\ A_{J2} & \varnothing \end{pmatrix} \tag{B.8}$$

$$A_{J2\,4,1} = \frac{-3J_2\mu R_\oplus^2}{2}\left[\left(1-\frac{5r_K^2}{r^2}\right)\left(\frac{1}{r^5}-\frac{5r_I^2}{r^7}\right)+\frac{10r_I^2r_K^2}{r^9}\right] \tag{B.9}$$

$$A_{J2\,4,2} = \frac{-3J_2\mu R_\oplus^2}{2} r_I\left[\left(1-\frac{5r_K^2}{r^2}\right)\left(-\frac{5r_J}{r^7}\right)+\frac{10r_Jr_K^2}{r^9}\right] \tag{B.10}$$

$$A_{J2\,4,3} = \frac{-3J_2\mu R_\oplus^2}{2} r_I\left[\left(1-\frac{5r_K^2}{r^2}\right)\left(-\frac{5r_K}{r^7}\right)+\left(\frac{10r_K}{r^7}\right)\left(\frac{r_K^2}{r^2}-1\right)\right] \tag{B.11}$$

$$A_{J2\,5,1} = \frac{-3J_2\mu R_\oplus^2}{2} r_J\left[\left(1-\frac{5r_K^2}{r^2}\right)\left(-\frac{5r_I}{r^7}\right)+\frac{10r_Ir_K^2}{r^9}\right] \tag{B.12}$$

$$A_{J2\,5,2} = \frac{-3J_2\mu R_\oplus^2}{2}\left[\left(1-\frac{5r_K^2}{r^2}\right)\left(\frac{1}{r^5}-\frac{5r_J^2}{r^7}\right)+\frac{10r_J^2r_K^2}{r^9}\right] \tag{B.13}$$

$$A_{J2\,5,3} = \frac{-3J_2\mu R_\oplus^2}{2} r_J\left[\left(1-\frac{5r_K^2}{r^2}\right)\left(-\frac{5r_K}{r^7}\right)+\left(\frac{10r_K}{r^7}\right)\left(\frac{r_K^2}{r^2}-1\right)\right] \tag{B.14}$$

$$A_{J2\,6,1} = \frac{-3J_2\mu R_\oplus^2}{2} r_K\left[\left(3-\frac{5r_K^2}{r^2}\right)\left(-\frac{5r_I}{r^7}\right)+\frac{10r_Ir_K^2}{r^9}\right] \tag{B.15}$$

$$A_{J2\,6,2} = \frac{-3J_2\mu R_\oplus^2}{2} r_K \left[\left(3 - \frac{5r_K^2}{r^2} \right) \left(-\frac{5r_J}{r^7} \right) + \frac{10 r_J r_K^2}{r^9} \right] \tag{B.16}$$

$$A_{J2\,6,3} = \frac{-3J_2\mu R_\oplus^2}{2} \left[\left(3 - \frac{5r_K^2}{r^2} \right) \left(\frac{1}{r^5} - \frac{5r_K^2}{r^7} \right) + \left(\frac{10 r_K^2}{r^7} \right) \left(\frac{r_K^2}{r^2} - 1 \right) \right] \tag{B.17}$$

Equations of Variation for 3rd Body perturbations

Recall that $f_{3-Body}(X) = a_{3-Body} = \mu_{3rdBody} \left(\frac{\bar{r}_{3rdBody\,to\,Sat}}{r_{3rdBody\,to\,Sat}^3} - \frac{\bar{r}_{3rdBody\,to\,Earth}}{r_{3rdBody\,to\,Earth}^3} \right)$ (B.18)

$$A_{rr\ 3-Body} = \nabla f_{3-Body} = \begin{pmatrix} \varnothing & \varnothing \\ A_{3-Body} & \varnothing \end{pmatrix} \tag{B.19}$$

$$A_{3-Body\ 4,1} = -\mu_{3rdBody} \left(\frac{1}{r_{3rdBody\,to\,Sat}^3} - 3\frac{r_I^2}{r_{3rdBody\,to\,Sat}^5} \right) \tag{B.20}$$

$$A_{3-Body\ 4,2} = \mu_{3rdBody} \frac{3 r_I r_J}{r_{3rdBody\,to\,Sat}^5} \tag{B.21}$$

$$A_{3-Body\ 4,3} = \mu_{3rdBody} \frac{3 r_I r_K}{r_{3rdBody\,to\,Sat}^5} \tag{B.22}$$

$$A_{3-Body\ 5,1} = \mu_{3rdBody} \frac{3 r_I r_J}{r_{3rdBody\,to\,Sat}^5} \tag{B.23}$$

$$A_{3-Body\ 5,2} = -\mu_{3rdBody} \left(\frac{1}{r_{3rdBody\,to\,Sat}^3} - 3\frac{r_J^2}{r_{3rdBody\,to\,Sat}^5} \right) \tag{B.24}$$

$$A_{3-Body\ 5,3} = \mu_{3rdBody} \frac{3 r_J r_K}{r_{3rdBody\,to\,Sat}^5} \tag{B.25}$$

$$A_{3-Body\ 6,1} = \mu_{3rdBody} \frac{3 r_I r_K}{r_{3rdBody\,to\,Sat}^5} \tag{B.26}$$

$$A_{3-Body\ 6,2} = \mu_{3rdBody} \frac{3 r_J r_K}{r_{3rdBody\,to\,Sat}^5} \tag{B.27}$$

$$A_{3-Body\ 6,3} = -\mu_{3rdBody} \left(\frac{1}{r_{3rdBody\,to\,Sat}^3} - 3 \frac{r_K^2}{r_{3rdBody\,to\,Sat}^5} \right) \tag{B.28}$$

Equations of Variation for Drag

Recall that $f_{drag}(X) = a_{drag} = -\frac{1}{2} \rho \frac{C_D A}{m} v_{rel}^2 \frac{\vec{v}_{rel}}{|\vec{v}_{rel}|}$ (B.29)

$$A_{rr,rv\ drag} = \nabla f_{drag} = \begin{pmatrix} \varnothing & \varnothing \\ A_{drag\ rr} & A_{drag\ rv} \end{pmatrix} \tag{B.30}$$

$$A_{drag\ 4,1} = -\frac{1}{2} \rho \frac{C_D A}{m} (v_I + \omega_\oplus r_J) \left[-\frac{r_I v_{rel}}{Hr} - \frac{\omega_\oplus (v_J - \omega_\oplus r_I)}{v_{rel}} \right] \tag{B.31}$$

$$A_{drag\ 4,2} = -\frac{1}{2} \rho \frac{C_D A}{m} \left[-\frac{r_J v_{rel} (v_I + \omega_\oplus r_J)}{Hr} + \frac{\omega_\oplus (v_I + \omega_\oplus r_J)^2}{v_{rel}} + v_{rel} \omega_\oplus \right] \tag{B.32}$$

$$A_{drag\ 4,3} = -\frac{1}{2} \rho \frac{C_D A}{m} \left[-\frac{r_K v_{rel} (v_I + \omega_\oplus r_J)}{Hr} \right] \tag{B.33}$$

$$A_{drag\ 4,4} = -\frac{1}{2} \rho \frac{C_D A}{m} \left[\frac{(v_I + \omega_\oplus r_J)^2}{v_{rel}} + v_{rel} \right] \tag{B.34}$$

$$A_{drag\ 4,5} = -\frac{1}{2} \rho \frac{C_D A}{m} \left[\frac{(v_I + \omega_\oplus r_J)(v_J - \omega_\oplus r_I)}{v_{rel}} \right] \tag{B.35}$$

$$A_{drag\ 4,6} = -\frac{1}{2} \rho \frac{C_D A}{m} \left[\frac{v_K (v_I + \omega_\oplus r_J)}{v_{rel}} \right] \tag{B.36}$$

$$A_{drag\ 5,1} = -\frac{1}{2} \rho \frac{C_D A}{m} \left[-\frac{r_I v_{rel} (v_J - \omega_\oplus r_I)}{Hr} - \frac{\omega_\oplus (v_J + \omega_\oplus r_I)^2}{v_{rel}} - v_{rel} \omega_\oplus \right] \tag{B.37}$$

$$A_{drag\ 5,2} = -\frac{1}{2}\rho\frac{C_D A}{m}(v_J - \omega_\oplus r_I)\left[-\frac{r_J v_{rel}}{Hr} + \frac{\omega_\oplus (v_I + \omega_\oplus r_J)}{v_{rel}}\right] \tag{B.38}$$

$$A_{drag\ 5,3} = -\frac{1}{2}\rho\frac{C_D A}{m}\left[-\frac{r_K v_{rel}(v_J - \omega_\oplus r_I)}{Hr}\right] \tag{B.39}$$

$$A_{drag\ 5,4} = -\frac{1}{2}\rho\frac{C_D A}{m}\left[\frac{(v_I + \omega_\oplus r_J)(v_J - \omega_\oplus r_I)}{v_{rel}}\right] \tag{B.40}$$

$$A_{drag\ 5,5} = -\frac{1}{2}\rho\frac{C_D A}{m}\left[\frac{(v_J - \omega_\oplus r_I)^2}{v_{rel}} + v_{rel}\right] \tag{B.41}$$

$$A_{drag\ 5,6} = -\frac{1}{2}\rho\frac{C_D A}{m}\left[\frac{v_K (v_J - \omega_\oplus r_I)}{v_{rel}}\right] \tag{B.42}$$

$$A_{drag\ 6,1} = -\frac{1}{2}\rho\frac{C_D A}{m}v_K\left[-\frac{r_I v_{rel}}{Hr} - \frac{\omega_\oplus (v_J - \omega_\oplus r_I)}{v_{rel}}\right] \tag{B.43}$$

$$A_{drag\ 6,2} = -\frac{1}{2}\rho\frac{C_D A}{m}v_K\left[-\frac{r_J v_{rel}}{Hr} + \frac{\omega_\oplus (v_I + \omega_\oplus r_J)}{v_{rel}}\right] \tag{B.44}$$

$$A_{drag\ 6,3} = -\frac{1}{2}\rho\frac{C_D A}{m}v_K\left[-\frac{r_K v_{rel}}{Hr}\right] \tag{B.45}$$

$$A_{drag\ 6,4} = -\frac{1}{2}\rho\frac{C_D A}{m}\left[\frac{v_K (v_I + \omega_\oplus r_J)}{v_{rel}}\right] \tag{B.46}$$

$$A_{drag\ 6,5} = -\frac{1}{2}\rho\frac{C_D A}{m}\left[\frac{v_K (v_J - \omega_\oplus r_I)}{v_{rel}}\right] \tag{B.47}$$

$$A_{drag\ 6,6} = -\frac{1}{2}\rho\frac{C_D A}{m}\left[\frac{v_K^2}{v_{rel}} + v_{rel}\right] \tag{B.48}$$

Appendix C: Description of the Estimation Process as Coded in MATLAB

The MATLAB Bayes filter is one file in a folder of over a dozen routines required to actualize a simulation, including the Non-Linear Least Squares filter. The following five steps describe how an estimate is generated for a given scenario in MATLAB:

==========

Step 1 - Establish data files

==========

To build a data set: Run "data generator."

(Programming Note - If 'data_generator' is run for a single site's observations, then the MATLAB 'w+' flag should be used in the routine's "file-create-and-write" calls to ensure any previous data is overwritten. If data must be generated for consecutive observations all to be processed at one time, a separate 'data_generator' routine should be set up for each site to be run consecutively (in chronological order), with the MATLAB 'a' flag ("append") used in the routine for the second and all following sites' "file-create-and-write" calls to ensure subsequent observation data are appended to the full data set.)

The following data must be put into the data generator:

- an initial "truth" state with Julian date (say from SOAP)
- an observation site's location (lat, long, elevation)
- satellite mass, drag area, drag coefficient
- time step and number of data points

(say 2 sec step for 90 points gives 3 minutes of data)

- sensor instrument Sigmas. Sigmas are multipliers for random numbers generated to introduce error in observation data elements.

Data generator calls the following functions:

1. integration routine call: '@rhs' is the function containing the equations to be integrated (2-body motion and perturbations. 'time_vec' is the time span to be integrated over. X is the current state of the system (initial conditions).
2. subroutine 'lstime' to get Local Sidereal Time for the tracking site. (Greenwich Sidereal Time, GST, not used). LST in degrees. (Vallado routine)
3. subroutine 'site' to get instantaneous tracking site position vector in ECI coordinates according to the Julian Date. Units are in kilometers. (Vallado routine)
4. subroutine 'razel' to generate range/azimuth/elevation data (Vallado routine)
5. subroutine 'topocentric' to generate right ascension and declination data. (Vallado routine)

Data generator creates and populates the following files:

- 'range_az_el_data.txt'
- 'range_and_range_rate_data.txt'
- 'azimuth_and_elevation_data.txt'
- 'rt_asc_and_declination_data.txt'
- 'range_rate_only_data.txt'
- 'tracking_site_position_data.txt'
- 'satellite_position_velocity_data.txt'

==========

Step 2 - Establish Observation vectors from site

==========

Once data has been generated, build a data set of simulated observation vectors by running "Gibbs_vectors.m"

The following data must be put into Gibbs_vectors:

- range rate
- azimuth rate
- elevation rate

(All assumed to be 0 for these observations)

Gibbs_vectors reads the following functions:

1. Datafile 'range_az_el_data.txt' created by the data generator (with associated instrument sigmas
2. subroutine ‘site_track’ returns observation site vector in ECI (inertial) coordinate system, Satellite absolute position and velocity in the ECI (inertial) IJK coordinate system. (r_ijk, v_ijk) Units in km and km/s. (Vallado routine)
3. subroutine 'lstime' to get Local Sidereal Time for the tracking site. LST in degrees. (Vallado routine)

Gibbs_vectors populates the following files:

- 'Gibbs_vectors_output.txt'

(Note - Only One output file is generated. Separate additional files need to be generated if observation data from more than one site needs to be generated for initial orbit determination and stored)

==========

Step 3 - Establish Initial Orbit State

==========

To establish an initial orbit state, run either function 'gibbs.m' or 'h_gibbs. m' (both are Vallado routines). Use Gibbs if the delta-nu is above 5 degrees. Use Herrick-Gibbs if the delta-nu is below 1degree. Use either (equal performance) if delta-nu is between 1 and 5 degrees. The routines test for delta-nu and advise to switch routines if the delta-nu is sub-optimal.

1. manually choose three vectors from the data file 'gibbs_vectors_output.txt'
2. ‘gibbs.m’ and ‘h_gibbs.m’ calculate the velocity vector associated with the second position vector,
3. with a range and associated velocity vector, the orbit is considered "determined,” thus a reference state has been established to initialize the non-linear least squares routine

==========

Step 4 - Estimate Orbit State using Non-linear Least Squares

==========

To initially estimate the orbit state, run ‘non_linear_lsq_filter’

The following data must be put into the non linear least squares filter:

- data type being evaluated (choose 1 through 8; e.g. type 1 is radar range/az/el data)
- reference orbit estimate x=(r,v) from gibbs or h_gibbs along with associated Julian Date
- processing flags for any perturbations to be modeled. (3-body, drag, J2)

0 = perturbation turned off

1 = perturbation turned on

- Satellite parameters for use in calculating drag: mass, drag area, drag coefficient. These should match what was used to generate data in step 1.
- "Mode" value is the flag for deciding whether the equations of variation are processed in subroutine '@ground_rhs' which provides the differential equations to be integrated.

'non_linear_lsq_filter' reads the following functions:

1. observations from data file based on data type (1 though 8).
   The files read here are created by the 'data_generator' program. They would otherwise come from real observations from a real sensor.
2. @ground_rhs is the function containing the equations to be integrated. 'time_vec' is the time span to be integrated over
3. subroutine 'lstime' to get Local Sidereal Time, LST in degrees which is used in subroutine 'obser' for calculating tracking site inertial position at time = current/observation JD
4. subroutine 'obser' to get predicted data vector, zpred, H matrix, Q_inv matrix. The Q matrix in this routine contains the instrument sigmas for the observation data. If a pseudo-state is used, this routine calculates the dynamic Q matrix based on Wiesel's Jacobian.

'non_linear_lsq_filter' creates and populates the following files:

- 'range_az_el_residuals_output.txt'

- 'range_and_range_rate_residuals_output.txt'

- 'az_el_residuals_output.txt'

- 'rt_asc_and_dec_residuals_output.txt'

- 'range_rate_residuals_output.txt'

- 'sat_position_velocity_output.txt'

- 'state_and_state_corrections_output.txt'

- 'covariance_matrix.txt'

‘non_linear_lsq_filter’ prints to the screen:

- status of estimate

- residual vector,

- P matrix, State matrix X, correction delta-X, etc.

‘non_linear_lsq_filter’ plots:

- For each data element (say range, or azimuth, or elevation), 2 figures:

(1) observed vs predicted data over the collection period

(2) first pass and last pass residuals vs observed data over the collection period

==========

Step 5 - Estimate Orbit State using Bayes

==========

To sequentially update the orbit state estimate, run ‘bayes_filter’

The following data must be put into the non linear least squares filter:

- data type being evaluated (choose 1 through 8; e.g. type 1 is radar range/az/el data)

- reference orbit estimate along with associated Julian Date. This is the estimate established by the non-linear least squares filter
- covariance matrix P_minus established by the non-linear least squares filter
- processing flags for any perturbations to be modeled. (3-body, drag, J2)

  0 = perturbation turned off

  1 = perturbation turned on
- Satellite parameters for use in calculating drag: mass, drag area, drag coefficient. These should match what was used to generate data in step 1 as well as what was used by the non linear least squares routine in step 4.
- "Mode" value is the flag for deciding whether the equations of variation are processed in subroutine '@ground_rhs' which provides the differential equations to be integrated.

Bayes_filter reads the following files and functions:

1. 'estimate_and_epoch_output.txt' generated by last filter run (either Linear Least Squares or Bayes. IMPORTANT: ensure this file is copied to a separate directory to use again in future runs, since the file is overwritten as soon as its read for the current filter's output.
2. 'covariance_matrix.txt' generated by last filter run (either Linear Least Squares or Bayes. IMPORTANT: ensure this file is copied to a separate directory to use again in future runs, since the file is overwritten as soon as its read for the current filter's output.
3. observations from data file based on data type (1 though 8).

The files read here are created by the 'data_generator' program. They would otherwise come from real observations from a real sensor.

3. @ground_rhs is the function containing the equations to be integrated. 'time_vec' is the time span to be integrated over
4. subroutine 'lstime' to get Local Sidereal Time, LST in degrees which is used in subroutine 'obser' for calculating tracking site inertial position at time = current/observation JD
5. subroutine 'obser' to get predicted data vector, zpred, H matrix, Q_inv matrix. The Q matrix in this routine contains the instrument sigmas for the observation data. If a pseudo-state is used, this routine calculates the dynamic Q matrix based on Wiesel's Jacobian.

Bayes_filter creates and populates the following files:

- 'range_az_el_residuals_output.txt'
- 'range_and_range_rate_residuals_output.txt'
- 'az_el_residuals_output.txt'
- 'rt_asc_and_dec_residuals_output.txt'
- 'range_rate_residuals_output.txt'
- 'sat_position_velocity_output.txt'
- 'state_and_state_corrections_output.txt'
- 'covariance_matrix.txt'

Bayes_filter prints to the screen:

- status of estimate

- P matrix, State matrix X, etc

Bayes_filter plots:

- For each data element, 2 figures:

  (1) observed vs predicted data over the collection period

  (2) first pass and last pass residuals vs observed data over the collection period

Appendix D: Bayes Orbit Determination Filter MATLAB Code

```
% LtCol John Heslin
% Bayes filter
% 27 December 2004
% Adapted from Wiesel

clear all
close all

format long g

% Open output data files.
% Start by copying the output files from the old estimate run into the same
% directory as the Bayes Filter m file

% Read in
% 1. old estimate with associated JD (X_old) and,
% 2. old estimate's associated covariance (P_old)

% All subsequent observation (X_minus) and covariances (P_minus) will come
% from the files written after each successive ground station's data are
% processed

% Bring the old estimate and covariance up to the current epoch
% P_minus = Phi*P_old*Phi'

% 3. new data (from new site(?)) with associated JD.  Again, for the first
% time through, be sure to run the first ground stations "data generator"
% to write the necessary files

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Bayes -
% Read in old X estimate and epoch here from non-lin least square filter
%
[old_estimate_file] = textread('estimate_and_epoch_output.txt',','delimiter',' ');
      % returns a matrix that is of dimensions
      % (# of times steps x # of equations integrated)
      old_estimate_file_size = size(old_estimate_file);
      old_estimate_file_length = length(old_estimate_file);
      old_estimate_file_last_row = old_estimate_file_size(1);

      % Extract only the last time step values of state X
      % because ode45 expects a 42 component column vector.
```

```
    old_estimate = old_estimate_file(old_estimate_file_last_row,:);

    JD_old = old_estimate(1)
    X_old = old_estimate(2:7)'

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Bayes -
% Read in old P covariance matrix
%
[P_old] = textread('covariance_matrix.txt','','delimiter',' ');
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Open output data files.
fid1 = fopen('range_az_el_residuals_output.txt','w+'); % radar
fid2 = fopen('range_and_range_rate_residuals_output.txt','w+');
fid3 = fopen('az_el_residuals_output.txt','w+'); %optical
fid4 = fopen('rt_asc_and_dec_residuals_output.txt','w+'); %optical
fid5 = fopen('range_rate_residuals_output.txt','w+');
fid6 = fopen('sat_position_velocity_output.txt','w+');
fid7 = fopen('state_and_state_corrections_output.txt','w+');
fid8 = fopen('covariance_matrix.txt','w+');
fid9 = fopen('estimate_and_epoch_output.txt','w+');
fid10 = fopen('phi_matrix.txt','w+');
fid11 = fopen('state_residuals_output.txt','w+');
fid12 = fopen('state_range_residuals_output.txt','w+');


% Select type of data that will be processed on this filter run.
% Type 1: range, azimuth angle, and elevation angle
% Type 2: range and range-rate
% Type 3: azimuth angle and elevation angle
% Type 4: right ascension angle and declination angle
% Type 5: range-rate only
% Type 7: state vector (for troubleshooting)
% Type 8: pseudo-state range vector

data_type = 8;  %

% X(1) = I component of position vector r in the IJK coordinate system
% X(2) = J component of position vector r in the IJK coordinate system
% X(3) = K component of position vector r in the IJK coordinate system
% X(4) = I component of velocity vector v in the IJK coordinate system
% X(5) = J component of velocity vector v in the IJK coordinate system
% X(6) = K component of velocity vector v in the IJK coordinate system
```

```
% JD_epoch = 2452340.332639; %   < == grabbed from SOAP
% % 6 March 2002 1959hrs 00 sec
%
% X_ref(1) = -5825.5628;
% X_ref(2) = -2173.4191;
% X_ref(3) = 2927.4442;
% X_ref(4) = 3.28502695;
% X_ref(5) = -6.69629618;
% X_ref(6) = 1.64518455;

% JD = 2452340.472917; %   < == grabbed from SOAP for China radar
%
% JD_epoch = JD
%
% X_ref(1) =  -2109.3158;
% X_ref(2) = -5785.9666;
% X_ref(3) = 3119.6134;
% X_ref(4) = 6.82539645;
% X_ref(5) = -3.15651721;
% X_ref(6) = -1.12407815;

JD = 2452340.512153; %   < == grabbed from SOAP for Brazil 1

JD_epoch = JD

X_ref(1) = -1519.3346;
X_ref(2) = 6389.5532;
X_ref(3) = -2069.0750;
X_ref(4) = -7.03175959;
X_ref(5) = -0.80277910;
X_ref(6) = 2.83486064;

% JD = 2452340.516319; %   < == grabbed from SOAP for Brazil 2
%
% JD_epoch = JD
%
% X_ref(1) = -3865.2675;
% X_ref(2) = 5607.7414;
% X_ref(3) =  -912.9388;
% X_ref(4) = -5.82682292;
% X_ref(5) =  -3.48593603;
% X_ref(6) =  3.50340532;
```

```
% Put the reference trajectory into a column vector format.
X_ref = X_ref';

X_nom = [X_ref(1); X_ref(2); X_ref(3); X_ref(4); X_ref(5); X_ref(6)];

% Initialize the state corrections to 0 for
% writing to first line of output file.
del_X = [0; 0; 0; 0; 0; 0; 0];

% Initialize iteration to 0 for
% writing to output file.
iteration = 0;

% Write initial state and other data to output file.
fprintf(fid7,'%3d %15.6f %15.6f %15.6f %15.6f %15.6f %15.6f %15.6f %15.6f
%15.6f %15.6f %15.6f\n',...
   iteration,X_ref(1),X_ref(2),X_ref(3),X_ref(4),X_ref(5),X_ref(6),...
   del_X(1),del_X(2),del_X(3),del_X(4),del_X(5),del_X(6));


% Set processing flags for any perturbations to be modeled.
% 0 = perturbation turned off
% 1 = perturbation turned on

% Third gravitational effects from Sun and Moon
third_body_flag = 1;

% Earth's J2 zonal gravity harmonic due to oblateness
J2_flag = 1;

% Atmospheric drag
drag_flag = 1;

% Satellite parameters for use in calculating drag.

% Shuttle mass is 171,000 lbs
sat_mass = 77565; % kilograms

% Satellite cross-sectional area, in square meters
% Shuttle Drag Area : 2750.0 SQ FT = 255.48336 m^2

sat_area = 255.48336; % square meters

% Drag coefficient (dimensionless)
```

```
drag_coefficient = 1.0;
% This should be 2 according to the data for STS-61 found on
% on http://spacelink.nasa.gov/.index.html

%debugging
% new_X_size = 0;

%%%%%%%%%%%%%%%%%%%%%%%%%%
% Read in observations from data file based on data type.
% The files read here are created by the 'data_generator_all_types'
% program.  Or would otherwise come from real observations from
% a real sensor.

% Data for range, azimuth, and elevation.
% Can be converted to r,v with 'site_track' and 'gibbs' algorithms for
% X_nom(iob), but load 'satellite_position_velocity_data.txt' created by
% the data generator for the purposes of this simulation
if(data_type == 1)
load('range_az_el_data.txt','-ascii')
[ob_type,order_ob,JDay,ob_time,range,az,el,latitude,longitude,elevation]...
   =textread('range_az_el_data.txt','%d %d %f %f %f %f %f %f %f %f',-1);
end

% Data for range and range-rate
if(data_type == 2)
load('range_and_range_rate_data.txt','-ascii')
[ob_type,order_ob,JDay,ob_time,range,range_rate,latitude,longitude,elevation]...
   =textread('range_and_range_rate_data.txt','%d %d %f %f %f %f %f %f %f',-1);
end

% Data for azimuth and elevation.
% Can be converted to r,v with 'azel_to_radec' and 'angles_only' algorithms
% for X_nom(iob), but load 'satellite_position_velocity_data.txt' created by
% the data generator for the purposes of this simulation.
if(data_type == 3)
load('azimuth_and_elevation_data.txt','-ascii')
[ob_type,order_ob,JDay,ob_time,az,el,latitude,longitude,elevation]...
   =textread('azimuth_and_elevation_data.txt','%d %d %f %f %f %f %f %f %f',-1);
end

% Data for topocentric right ascension and declination
if(data_type == 4)
load('rt_asc_and_declination_data.txt','-ascii')
[ob_type,order_ob,JDay,ob_time,ra,dec,latitude,longitude,elevation]...
```

```
   =textread('rt_asc_and_declination_data.txt','%d %d %f %f %f %f %f %f %f',-1);
end

% Data for range-rate
if(data_type == 5)
load('range_rate_only_data.txt','-ascii')
[ob_type,order_ob,JDay,ob_time,range_rate,latitude,longitude,elevation]...
   =textread('range_rate_only_data.txt','%d %d %f %f %f %f %f %f',-1);
end

% Data for full state vector
if(data_type == 7)
load('satellite_position_velocity_data.txt','-ascii')
[ob_type,order_ob,JDay,ob_time,r_i,r_j,r_k,v_i,v_j,v_k,latitude,...
   longitude,elevation]...
   =textread('satellite_position_velocity_data.txt','%d %d %f %f %f %f %f %f %f %f %f
%f %f',-1);
end

% Data for pseudo-state
if(data_type == 8)
load('satellite_position_only_data.txt','-ascii')
[ob_type,order_ob,JDay,ob_time,r_i,r_j,r_k,latitude,longitude,elevation]...
   =textread('satellite_position_only_data.txt','%d %d %f %f %f %f %f %f %f %f',-1);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Determine the number of observations which will determine the number
% of times through the data processing loop.
num_obs = length(ob_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Set up z, the total data (observation) vector.
% The "order" of z is the number of "types" of data
% associated with a single observation time.  For example,
% if processing range and range-rate then the order is 2.
% The dimension of z is (number of obs) x (order).
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Observation data type 1 is for range in kilometers and
% azimuth and elevation in degrees.
if (ob_type(1) == 1)
```

```
   z = [range az el];
end

% Observation data type 2 is for range in kilometers and
% range-rate in km/s.
if (ob_type(1) == 2)
   z = [range range_rate];
end

% Observation data type 3 is for azimuth and elevation
% in degrees.
if (ob_type(1) == 3)
   z = [az el];
end

% Observation data type 4 is for topocentric right
% ascension and declination in degrees.
if (ob_type(1) == 4)
   z = [ra dec];
end

% Observation data type 5 is for range-rate only in km/s.
if (ob_type(1) == 5)
   z = [range_rate];
end

% Observation data type 7 is for the state vector.
if (ob_type(1) == 7)

   z = [r_i r_j r_k v_i v_j v_k ];

end

% Observation data type 8 is for the pseudo-state vector.
if (ob_type(1) == 8)

      z = [r_i r_j r_k];

end
% Determine size of the z vector for debugging.
z_size = size(z);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Bring the old estimate and covariance matrix forward to the current epoch
```

```
% Get Phi by calling ode45 routine and integrating it forward over
% timestep equal to new JD_epoch - old JD, reading X_minus, and
% extracting Phi, then multiplying it through P

   phi = eye(6);

   % Initial "total" state conditions.  Reference trajectory and
   % the state transition matrix as a 42 x 1 column vector
   % since 'ode45' expects a column vector.
   X = [X_old;...
         phi(1,1); phi(1,2); phi(1,3); phi(1,4); phi(1,5); phi(1,6);...
         phi(2,1); phi(2,2); phi(2,3); phi(2,4); phi(2,5); phi(2,6);...
         phi(3,1); phi(3,2); phi(3,3); phi(3,4); phi(3,5); phi(3,6);...
         phi(4,1); phi(4,2); phi(4,3); phi(4,4); phi(4,5); phi(4,6);...
         phi(5,1); phi(5,2); phi(5,3); phi(5,4); phi(5,5); phi(5,6);...
         phi(6,1); phi(6,2); phi(6,3); phi(6,4); phi(6,5); phi(6,6)];

   % Verify X is a 42 x 1 column vector for debugging.
  X_size = size(X);
  time_step = (JD_epoch - JD_old)*86400.0;
  time_vec = 0:time_step;

      abs_tol = 1e-8* ones(42,1);

      options = odeset('RelTol', 1e-8, 'AbsTol', abs_tol);

      % ode45 is one of MATLAB's built-in numerical integrators.  It is
      % based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince
      % pair.  It is a one-step solver in computing X(t), it needs only
      % the solution at the immediately preceding time point, X(t n-1).

      % Format of the integration routine call:
      %   @ground_rhs is the function containing the equations to be integrated.
      %   time_vec is the time span to be integrated over.
      %   X is the current state of the system  (initial conditions).
      %   options contain the information for absolute/relative
      %   tolerances, etc.
      iob = 0;
      mode = 1;
      [t,X_old_propagated] = ode45(@ground_rhs,time_vec,X,options,mode,JD,...
         third_body_flag,J2_flag, drag_flag,drag_coefficient,...
         sat_mass,sat_area,iob);

      % ode45 returns a matrix that is of dimensions
```

```
        % (# of times steps x # of equations integrated)
        X_old_propagated_size = size(X_old_propagated);
        X_old_propagated_length = length(X_old_propagated);
        X_old_propagated_last_row = X_old_propagated_size(1);

        % Extract only the last time step values of state X
        X_minus = X_old_propagated(X_old_propagated_last_row,1:6);
        % Extract phi matrix in normal form from the total "state"
        % column vector X_calc calculated by ground_rhs

 phi = [X_old_propagated(7) X_old_propagated(8) X_old_propagated(9)...
     X_old_propagated(10) X_old_propagated(11) X_old_propagated(12);
       X_old_propagated(13) X_old_propagated(14) X_old_propagated(15)...
       X_old_propagated(16) X_old_propagated(17) X_old_propagated(18);
       X_old_propagated(19) X_old_propagated(20) X_old_propagated(21)...
       X_old_propagated(22) X_old_propagated(23) X_old_propagated(24);
       X_old_propagated(25) X_old_propagated(26) X_old_propagated(27)...
       X_old_propagated(28) X_old_propagated(29) X_old_propagated(30);
       X_old_propagated(31) X_old_propagated(32) X_old_propagated(33)...
       X_old_propagated(34) X_old_propagated(35) X_old_propagated(36);
       X_old_propagated(37) X_old_propagated(38) X_old_propagated(39)...
       X_old_propagated(40) X_old_propagated(41) X_old_propagated(42)];

       Phi_size = size(phi);

% Bring the old covariance up to the current epoch
P_minus = phi*P_old*phi';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% BAYES X_ref equals  X_minus
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

X_ref = X_minus;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Put the reference trajectory into a column vector format.
% The old estimate is the reference trajectory

X_ref = [X_ref(1); X_ref(2); X_ref(3); X_ref(4); X_ref(5); X_ref(6)];

% Initialize the state corrections to 0 for
```

```
% writing to first line of output file.
del_X = [0; 0; 0; 0; 0; 0; 0];

% Initialize iteration to 0 for
% writing to output file.
iteration = 0;

% Write initial state and other data to output file.
fprintf(fid7,'%3d %15.6f %15.6f %15.6f %15.6f %15.6f %15.6f %15.6f %15.6f
%15.6f %15.6f %15.6f\n',...
   iteration,X_ref(1),X_ref(2),X_ref(3),X_ref(4),X_ref(5),X_ref(6),...
   del_X(1),del_X(2),del_X(3),del_X(4),del_X(5),del_X(6));

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Set maximum number of iterations.
max_iter = 100;

% Initialize iteration so the 'while' loop will begin processing.
iteration = 1;
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%  Begin iteration loop for Non-Linear Least Squares

while iteration <= max_iter

   % "Mode" value is the flag for deciding whether only the equations
   % of motion (EOM)(mode = 0) or EOM and equations of
   % variation (EOM + EOV)(mode = 1) are processed in subroutine "rhs"
   % which provides the differential equations to be integrated.

   mode = 1;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Initialize the "total" state vector.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   % Initialize the state transition matrix, phi,

   phi = eye(6);

   % Initial "total" state conditions.  Reference trajectory and
   % the state transition matrix as a 42 x 1 column vector
```

```
% since 'ode45' expects a column vector.
X = [X_ref;...
      phi(1,1); phi(1,2); phi(1,3); phi(1,4); phi(1,5); phi(1,6);...
      phi(2,1); phi(2,2); phi(2,3); phi(2,4); phi(2,5); phi(2,6);...
      phi(3,1); phi(3,2); phi(3,3); phi(3,4); phi(3,5); phi(3,6);...
      phi(4,1); phi(4,2); phi(4,3); phi(4,4); phi(4,5); phi(4,6);...
      phi(5,1); phi(5,2); phi(5,3); phi(5,4); phi(5,5); phi(5,6);...
      phi(6,1); phi(6,2); phi(6,3); phi(6,4); phi(6,5); phi(6,6)];

% Verify X is a 42 x 1 column vector for debugging.
X_size = size(X);

X_calc = X;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Initialize buffers for matrix product accumulation.
%
% The matrices used in this program are:
%   phi - state transition matrix (6 x 6)
%   H   - observation model (order_obs x 6)
%   T   - observation matrix; product of H * phi; (order_obs x 6)
%   Q   - instrument covariance matrix (order_obs x order_obs)
%   r   - residual vector
%   P   - state covariance matrix (6 x 6)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initialize state correction term, dx, to zero first
del_X = zeros(6,1);

% For product of (T transpose)(Q inverse)(r)
% Dimensions:     (6 x n)    (n x n)   (n x 1) = (6 x 1)
Tz_tran_Q_inv_r = zeros(6,1);
% Initialize new covariance matrix inverse Tz_tran_Q_inv_Tz (6 x 6)
Tz_tran_Q_inv_Tz = zeros(6,6);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Observation processing loop
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for iob = 1:num_obs
```

```
% Write iob value to screen for progress monitoring.
 fprintf('Iteration %d of %d\n',iteration,max_iter)
 fprintf('%d of %d observations is processing.\n',...
    iob,num_obs)
 fprintf('\n')
 % Blank line for spacing on screen.

 % Julian Date needed by function 'LST' for
% determining tracking site inertial position.
JD = JDay(iob);

% The "order" of the observation determines the
% number of rows in the observation z matrix.
ndata = order_ob(iob);

% Z matrix data type identifier to pass to function 'obser'
% so that proper section can process obs and return the
% predicted data, zpred, matrix and the H matrix.
ztype = ob_type(iob);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Numerically integrate state and state transition matrix
% derivatives to observation time.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Determine the time_vector for ode45 based on and ensure
% the dynamics are progressing forward in time

if (iob == 1)
    % time_vec = 0:ob_time(1);
    % time_vec = 0:300:(JDay(1) - JD_ref)*86400.0
    % time_vec = 0:time_step(iob); % read in with satellite r,v data
    time_step = (JDay(1) - JD_epoch)*86400.0
    time_vec = 0:time_step;
end

if(iob > 1)
    % time_step = ob_time(iob) - ob_time(iob-1);
   time_step = (JDay(iob) - JDay(iob-1))*86400.0;
   time_vec = 0:time_step;
end
```

```
check_forward_motion = time_step; % Are we going forward in time?

if check_forward_motion >= 1; %sec

abs_tol = 1e-8* ones(42,1);

options = odeset('RelTol', 1e-7, 'AbsTol', abs_tol);

% Propogate X_nom to find X_calc

% ode45 is one of MATLAB's built-in numerical integrators.  It is
% based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince
% pair.  It is a one-step solver in computing X(t), it needs only
% the solution at the immediately preceding time point, X(t n-1).

% Format of the integration routine call:
%   @rhs is the function containing the equations to be integrated.
%   time_vec is the time span to be integrated over.
%   X is the current state of the system  (initial conditions).
%   options contain the information for absolute/relative
%   tolerances, etc.

mode = 1;

[t,Y] = ode45(@ground_rhs,time_vec,X_calc,options,mode,JD,...
   third_body_flag,J2_flag, drag_flag,drag_coefficient,...
   sat_mass,sat_area,iob);

% ode45 returns a matrix that is of dimensions
% (# of times steps x # of equations integrated)
Y_ode_size = size(Y);
Y_length = length(Y);
last_row = Y_ode_size(1);

% Extract only the last time step values of state X
% because ode45 expects a 42 component column vector.
X_calc = Y(last_row,:)';

new_X_size = size(X_calc);

% Write satellite position and velocity to output file.
fprintf(fid6,'%15.5f %15.6f %15.6f %15.6f %15.6f %15.6f %15.6f\n',...
ob_time(iob),X_calc(1),X_calc(2),X_calc(3),X_calc(4),...
```

```
X_calc(5),X_calc(6));

long = longitude(iob);
lat = latitude(iob);
alt = elevation(iob);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Read Julian Date associated with this observation.
JD = JDay(iob);

% Call to subroutine 'lstime' to get Local Sidereal Time, LST
% in degrees which is used in subroutine 'obser' for
% calcualating tracking site inertial position.
[GST,LST] = lstime(JD,long);

% Call to subroutine 'obser' to get predicted data vector,
% zpred, H matrix, Q_inv matrix.
[zpred,H,Q_inv] = obser(X_calc,ztype,lat,alt,LST);

zpred_size = size(zpred);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Initialize the residual rejection flag.
rejected = 0;

if(ob_type(iob) == 1)
   r = [z(iob,1) - zpred(1);
        z(iob,2) - zpred(2);
        z(iob,3) - zpred(3)]
end


if((ob_type(iob) == 2) | (ob_type(iob) == 3) | (ob_type(iob) == 4))
   r = [z(iob,1) - zpred(1);
        z(iob,2) - zpred(2)]
end

if(ob_type(iob) == 5)
   r = [z(iob,1) - zpred(1)]
end


if(ob_type(iob) == 7)
   r = [z(iob,1) - zpred(1);
       z(iob,2) - zpred(2);
       z(iob,3) - zpred(3);
```

```
            z(iob,4) - zpred(4);
            z(iob,5) - zpred(5);
            z(iob,6) - zpred(6)];
        end

        if(ob_type(iob) == 8)
           r = [z(iob,1) - zpred(1);
            z(iob,2) - zpred(2);
            z(iob,3) - zpred(3)];
        end

        residual_vector_size = size(r);

% Write observed and predicted range, azimuth, and elevation
% and residuals to output file.
if(ob_type(iob) == 1)
   fprintf(fid1,'%15.5f %14.5f %14.8f %14.8f %14.8f %12.6f %12.6f %12.6f %11.6f
%11.6f %11.6f\n',...
        JD,ob_time(iob),z(iob,1),zpred(1),r(1),z(iob,2),zpred(2),...
        r(2),z(iob,3),zpred(3),r(3));

end

% Write observed and predicted range and range rate and
% residuals to output file.
if(ob_type(iob) == 2)
   fprintf(fid2,'%15.5f %14.5f %14.8f %14.8f %14.8f %10.6f %10.6f %10.6f\n',...
      JD,ob_time(iob),z(iob,1),zpred(1),r(1,1),z(iob,2),zpred(2),...
      r(2,1));
end

% Write observed and predicted azimuth and elevation and
% residuals to output file.
if(ob_type(iob) == 3)
   fprintf(fid3,'%15.5f %14.5f %10.6f %10.6f %10.6f %9.6f %9.6f %9.6f\n',...
      JD,ob_time(iob),z(iob,1),zpred(1),r(1),z(iob,2),zpred(2),...
      r(2));
end

% Write observed and predicted topocentric right ascension and
% declination and residuals to output file.
if(ob_type(iob) == 4)
   fprintf(fid4,'%15.5f %14.5f %10.6f %10.6f %10.6f %9.6f %9.6f %9.6f\n',...
      JD,ob_time(iob),z(iob,1),zpred(1),r(1),z(iob,2),zpred(2),r(2));
```

```
end

% Write observed and predicted range rate residuals to output file.
if(ob_type(iob) == 5)
   fprintf(fid5,'%15.5f %14.5f %10.6f %10.6f %10.6f\n',...
      JD,ob_time(iob),z(iob,1),zpred(1),r(1));
end

% Write observed and state
% and residuals to output file.
if(ob_type(iob) == 7)

   rz = [z(iob,1) z(iob,2) z(iob,3) ];
   rznorm = norm(rz);
   vz = [z(iob,4) z(iob,5) z(iob,6) ];
   vznorm = norm(vz);

   rp = [zpred(1) zpred(2) zpred(3) ];
   rpnorm = norm(rp);
   vp = [zpred(4) zpred(5) zpred(6) ];
   vpnorm = norm(vp);

   xres = rznorm - rpnorm;
   vres = vznorm - vpnorm;

   fprintf(fid11,'%15.5f %14.5f %14.8f %14.8f %14.8f %12.6f %12.6f %12.6f %11.6f
%11.6f %11.6f\n',...
      JD,ob_time(iob),rznorm,rpnorm,xres,vznorm,vpnorm,vres);

end

% write observed range and residuals to file
if(ob_type(iob) == 8)

   rz = [z(iob,1) z(iob,2) z(iob,3) ];
   rznorm = norm(rz);

   rp = [zpred(1) zpred(2) zpred(3) ];
   rpnorm = norm(rp);

   xres = rznorm - rpnorm;

   fprintf(fid12,'%15.5f %14.5f %14.8f %14.8f %14.8f %12.6f %12.6f %12.6f\n',...
      JD,ob_time(iob),rznorm,rpnorm,xres);
```

```
end
        reject = 300000.0;  % Wiesel did a number this large.

       for i = 1:ndata
          % Compare the elements of r(i) with its corresponding
          % diagonal entry of the Q_inv matrix.
          if(abs(r(i)) > reject/sqrt(Q_inv(i,i)))
             % Set residual rejection flag to sort/omit rejected obs.
             rejected = 1;
          end
       end  % End for i = 1:ndata


     % If the observation is not rejected, process its matrices.
     % Check if 'rejected' is anything other than 1 (not equal to 1).
        if (rejected ~= 1)

          % Extract phi matrix in normal form from the total "state"
          % column vector X_calc calculated by ground_rhs

      phi = [X_calc(7) X_calc(8) X_calc(9) X_calc(10) X_calc(11) X_calc(12);
           X_calc(13) X_calc(14) X_calc(15) X_calc(16) X_calc(17) X_calc(18);
           X_calc(19) X_calc(20) X_calc(21) X_calc(22) X_calc(23) X_calc(24);
           X_calc(25) X_calc(26) X_calc(27) X_calc(28) X_calc(29) X_calc(30);
           X_calc(31) X_calc(32) X_calc(33) X_calc(34) X_calc(35) X_calc(36);
           X_calc(37) X_calc(38) X_calc(39) X_calc(40) X_calc(41) X_calc(42)];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% For Bayes,
% H = dG/dX = I for old estimate
% T = [I ; Tz] ==> I for old data Tz bottom half for new data
% T'Q_inv = [I Tz] * [P(-) 0 ; 0  Q_inv]
% P(+)_inv = P(-)_inv + T'*Q_inv*T
% del_X = P(+)*T'*Q_inv*r = P(+) *[P(-)_inv , T'Q_inv]*[(x(-)-X_ref) ; r]
% P(+)_inv = P(-)_inv + T_tran*Q_inv*T
% P(-) is the covariance brought forward from the original estimate
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

          % Matrix dimension statements for debugging.
          % Remove ; at end of line to write to screen.
          H_size = size(H);
          phi_size = size(phi);
          P_minus_size = size(P_minus);
          Q_inv_size = size(Q_inv);
```

```
        [P_rows,P_cols] = size(P_minus);
        [Q_rows,Q_cols] = size(Q_inv);

        % Form matrix product  T    =    H   *  phi
        % Dimensions:       (n x 6) = (n x 6)*(6 x 6), where
        % n = ndata = order_ob
        % T is the observation matrix.
        Tz = H * phi;
        Tz_size = size(Tz);

        T  = [eye(P_minus_size);Tz];

        % Form Bayes product
        % P_inv = P_minus_inv + (T transpose)*(Q inverse)*(T)
        % This product is the "observability condition."  It
        % must be invertible for an estimate to exist.
        % Dimensions:   (6 x 6)= (6 x n)*(n x n)    *(n x 6)

        [P_rows,P_cols] = size(P_minus);
        [Q_rows,Q_cols] = size(Q_inv);
%           T_tran_Q_inv = T'*[P_minus      zeros(P_rows,Q_cols);
%                       zeros(Q_rows,P_cols)        Q_inv];

        % Matrix dimension statements for debugging.
        Q_inv_size = size(Q_inv);
        T_size = size(T);
        T_trans_size = size(T');
        r_size = size(r);

        % Form product (T transpose)*(Q inverse)*(T)
        % Dimensions: (6 x 1) = (6 x 1) + (6 x n)*(n x n)*(n x 1)
        Tz_tran_Q_inv_Tz = Tz_tran_Q_inv_Tz + (Tz' *Q_inv * Tz);

        % Form product (T transpose)*(Q inverse)*(r)
        % Dimensions: (6 x 1) = (6 x 1) + (6 x n)*(n x n)*(n x 1)
        Tz_tran_Q_inv_r = Tz_tran_Q_inv_r + (Tz' *Q_inv * r);

        end  % End to go with check of rejected ~= 1.

      rejected = 0;  % Reset rejected flag to 0 so that the next
                     % observation will be evaluated.

     end % End of loop for check_forward_motion logic check
```

```
end % End of for loop for iob = 1:num_obs

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   %
   % Data is processed; improve estimate of xref at EPOCH.
   % You are NOT calculating xref at every time step.
   %
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   % Bayes: P = (inv(P_minus) + T_tran_Q_inv * T )^-1
   P_inv = inv(P_minus) + Tz_tran_Q_inv_Tz; %Bayes

   % Invert matrix T transpose Q inverse T to find covariance P
   % Dimensions: (6 x 6) = inv((6 x n)*(n x n)*(n x 6))
   %         P = inv(T' * Q_inv * T);
     P = inv(P_inv);

   % Multiply P by T transpose Q inverse r to get correction
   % to the state vector.

   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   Vallado_Scale = (.05); % Make a correction that is
   % 5% of what is predicted by the residuals (note that Wiesel's
   % method gives a correction roughly equal to an average of the
   % residuals)
   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   % Dimensions: (6 x 1) = (6 x 6)*(6 x 1)

   X_difference = X_minus' - X_ref;

   del_X = (P*((inv(P_minus)*(X_difference)) +...
      Tz_tran_Q_inv_r)).*Vallado_Scale;

   % Establish the convergence criteria.

   this_close = .01; % .01 = 1%, .1 = 10%

   if((abs(del_X(1) > this_close*sqrt(abs(P(1,1)))))...
         | (abs(del_X(2) > this_close*sqrt(abs(P(2,2)))))...
         | (abs(del_X(3) > this_close*sqrt(abs(P(3,3)))))...
         | (abs(del_X(4) > this_close*sqrt(abs(P(4,4)))))...
         | (abs(del_X(5) > this_close*sqrt(abs(P(5,5)))))...
         | (abs(del_X(6) > this_close*sqrt(abs(P(6,6))))))
```

```
        % The vertical bar(s) in the above 'if' statement is/are
        % MATLAB's logical 'or' operator.

           convergence = 0;

        else
           convergence = 1;

      end

   del_X

   % Add in state corrections to reference state (trajectory)
   % This for the state at EPOCH only.  NOT every time step.

   X_ref = X_ref + del_X

   % Write current value of X_tgt_ref after corrections have been added.
   fprintf(fid7,'%3d %15.6f %15.6f %15.6f %15.6f %15.6f %15.6f %15.6f
%15.6f %15.6f %15.6f %15.6f\n',...
   iteration,X_ref(1),X_ref(2),X_ref(3),X_ref(4),X_ref(5),X_ref(6),...
   del_X(1),del_X(2),del_X(3),del_X(4),del_X(5),del_X(6));

   % Write the current JD as the epoch to use for future Bayes estimates
   fprintf(fid9,'%15.5f %15.6f %15.6f %15.6f %15.6f %15.6f %15.6f\n',...
      JD_epoch,X_ref(1),X_ref(2),X_ref(3),X_ref(4),X_ref(5),X_ref(6));

    if(convergence == 1)

       % Write iteration value to screen before increasing
       % and exiting loop.
       fprintf('Converged on iteration %d of %d\n',...
          iteration,max_iter);

       % Just add a number to get iterations to
       % exceed maximum and exit for loop;
       iteration = max_iter + 5;

    end

      % Increment iteration value by 1
      iteration = iteration + 1;
```

```
 end  % End statement for the iterations to max_iter loop

 % Write covariance matrix P to screen.
 P

 % Write the final covariance matrix components to output file here.
for i = 1:6
fprintf(fid8,'%25.20f %25.20f %25.20f %25.20f %25.20f %25.20f\n',...
   P(i,1),P(i,2),P(i,3),P(i,4),P(i,5),P(i,6));
end

% Principal error axes
% Extract the 3 x 3 space and velocity covariance submatrices

% Space covariance is the upper lefthand 3 x 3 of the P matrix
% Velocity covariance is the lower righthad 3 x 3 of the P matrix
for i = 1:3
   for j = 1:3
      space_P(i,j) = P(i,j);
      velocity_P(i,j) = P(i+3,j+3);
   end
end

% Eigenvector/value analysis of the covariance matrices
[V,D] = eig(space_P);
[W,E] = eig(velocity_P);

for i = 1:3
   raxis(i) = D(i,i);
   vaxis(i) = E(i,i);
end

for i = 1:3
   if(raxis(i) < 0.0)
      negative_r_axis = 'Space covariance has a negative value!'
   else
      raxis(i) = sqrt(raxis(i));
   end
end

for i = 1:3
   if(vaxis(i) < 0.0)
      negative_v_axis = 'Velocity covariance has negative value!'
   else
```

```
      vaxis(i) = sqrt(vaxis(i));
   end
end

raxis
vaxis

 % Close output files
 fclose(fid1);
 fclose(fid2);
 fclose(fid3);
 fclose(fid4);
 fclose(fid5);
 fclose(fid6);
 fclose(fid7);
 fclose(fid8);
 fclose(fid9);
 fclose(fid10);
 fclose(fid11);
 fclose(fid12);

 does_it_work = 'T-minus 15 days and counting!';

% Call plot function
% This version of 'plotted' gives the last pass residuals as it converged
[plotted] = plot_residuals(data_type,num_obs,iteration,JD_epoch)

% end of file
```

Appendix E: Non-Linear Least Squares Orbit Determination Filter MATLAB Code

```
% LtCol John Heslin
% Non-Linear Least Squares Filter
% 27 December 2004
% Adapted from Foster, Wiesel

clear all
close all

format long g

% Open output data files.
fid1 = fopen('range_az_el_residuals_output.txt','w+'); % radar
fid2 = fopen('range_and_range_rate_residuals_output.txt','w+');
fid3 = fopen('az_el_residuals_output.txt','w+'); %optical
fid4 = fopen('rt_asc_and_dec_residuals_output.txt','w+'); %optical
fid5 = fopen('range_rate_residuals_output.txt','w+');
fid6 = fopen('sat_position_velocity_output.txt','w+');
fid7 = fopen('state_and_state_corrections_output.txt','w+');
fid8 = fopen('covariance_matrix.txt','w+');
fid9 = fopen('estimate_and_epoch_output.txt','w+');
fid10 = fopen('phi_matrix.txt','w+');
fid11 = fopen('state_residuals_output.txt','w+');
fid12 = fopen('state_range_residuals_output.txt','w+');

% Select type of data that will be processed on this filter run.
% Type 1: range, azimuth angle, and elevation angle
% Type 2: range and range-rate
% Type 3: azimuth angle and elevation angle
% Type 4: right ascension angle and declination angle
% Type 5: range-rate only
% Type 7: full state vector r, r-dot
% Type 8: pseudo-range only

data_type = 8;  %

% X(1) = I component of position vector r in the IJK coordinate system
% X(2) = J component of position vector r in the IJK coordinate system
% X(3) = K component of position vector r in the IJK coordinate system
% X(4) = I component of velocity vector v in the IJK coordinate system
% X(5) = J component of velocity vector v in the IJK coordinate system
% X(6) = K component of velocity vector v in the IJK coordinate system
```

```
% JD_epoch = 2452340.332639; %   < == grabbed from SOAP
% % 6 March 2002 1959hrs 00 sec
%
% X_ref(1) = -5825.5628;
% X_ref(2) = -2173.4191;
% X_ref(3) = 2927.4442;
% X_ref(4) = 3.28502695;
% X_ref(5) = -6.69629618;
% X_ref(6) = 1.64518455;

% JD = 2452340.472917; %   < == grabbed from SOAP for China
%
% JD_epoch = JD
%
% X_ref(1) =  -2109.3158;
% X_ref(2) = -5785.9666;
% X_ref(3) = 3119.6134;
% X_ref(4) = 6.82539645;
% X_ref(5) = -3.15651721;
% X_ref(6) = -1.12407815;

% JD = 2452340.512153; %   < == grabbed from SOAP for Brazil 1
%
% JD_epoch = JD
%
% X_ref(1) = -1519.3346;
% X_ref(2) = 6389.5532;
% X_ref(3) = -2069.0750;
% X_ref(4) = -7.03175959;
% X_ref(5) = -0.80277910;
% X_ref(6) = 2.83486064;

JD = 2452340.516319; %   < == grabbed from SOAP for Brazil 2

JD_epoch = JD

X_ref(1) = -3865.2675;
X_ref(2) = 5607.7414;
X_ref(3) =  -912.9388;
X_ref(4) = -5.82682292;
X_ref(5) =  -3.48593603;
X_ref(6) =  3.50340532;

% Put the reference trajectory into a column vector format.
```

```
X_ref = X_ref';

X_nom = [X_ref(1); X_ref(2); X_ref(3); X_ref(4); X_ref(5); X_ref(6)];

% Initialize the state corrections to 0 for
% writing to first line of output file.
del_X = [0; 0; 0; 0; 0; 0; 0];

% Initialize iteration to 0 for
% writing to output file.
iteration = 0;

% Write initial state and other data to output file.
fprintf(fid7,'%3d %15.6f %15.6f %15.6f %15.6f %15.6f %15.6f %15.6f %15.6f
%15.6f %15.6f %15.6f\n',...
   iteration,X_ref(1),X_ref(2),X_ref(3),X_ref(4),X_ref(5),X_ref(6),...
   del_X(1),del_X(2),del_X(3),del_X(4),del_X(5),del_X(6));

% Set processing flags for any perturbations to be modeled.
% 0 = perturbation turned off
% 1 = perturbation turned on

% Third gravitational effects from Sun and Moon
third_body_flag = 1;

% Earth's J2 zonal gravity harmonic due to oblateness
J2_flag = 1;

% Atmospheric drag
drag_flag = 1;

% Satellite parameters for use in calculating drag.

% Shuttle mass is 171,000 lbs
sat_mass = 77565; % kilograms

% Satellite cross-sectional area, in square meters
% Shuttle Drag Area : 2750.0 SQ FT = 255.48336 m^2

sat_area = 255.48336; % square meters

% Drag coefficient (dimensionless)
drag_coefficient = 1.0;
% This should be 2 according to the data for STS-61 found on
```

```
% on http://spacelink.nasa.gov/.index.html

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Read in observations from data file based on data type.
% The files read here are created by the 'data_generator_all_types'
% program.  Or would otherwise come from real observations from
% a real sensor.

% Data for range, azimuth, and elevation.
% Can be converted to r,v with 'site_track' and 'gibbs' algorithms for
% X_nom(iob), but load 'satellite_position_velocity_data.txt' created by
% the data generator for the purposes of this simulation
if(data_type == 1)
load('range_az_el_data.txt','-ascii')
[ob_type,order_ob,JDay,ob_time,range,az,el,latitude,longitude,elevation]...
   =textread('range_az_el_data.txt','%d %d %f %f %f %f %f %f %f %f',-1);
end

% Data for range and range-rate
if(data_type == 2)
load('range_and_range_rate_data.txt','-ascii')
[ob_type,order_ob,JDay,ob_time,range,range_rate,latitude,longitude,elevation]...
   =textread('range_and_range_rate_data.txt','%d %d %f %f %f %f %f %f %f',-1);
end

% Data for azimuth and elevation.
% Can be converted to r,v with 'azel_to_radec' and 'angles_only' algorithms
% for X_nom(iob), but load 'satellite_position_velocity_data.txt' created by
% the data generator for the purposes of this simulation.
if(data_type == 3)
load('azimuth_and_elevation_data.txt','-ascii')
[ob_type,order_ob,JDay,ob_time,az,el,latitude,longitude,elevation]...
   =textread('azimuth_and_elevation_data.txt','%d %d %f %f %f %f %f %f %f',-1);
end

% Data for topocentric right ascension and declination
if(data_type == 4)
load('rt_asc_and_declination_data.txt','-ascii')
[ob_type,order_ob,JDay,ob_time,ra,dec,latitude,longitude,elevation]...
   =textread('rt_asc_and_declination_data.txt','%d %d %f %f %f %f %f %f %f',-1);
end

% Data for range-rate
if(data_type == 5)
```

```
load('range_rate_only_data.txt','-ascii')
[ob_type,order_ob,JDay,ob_time,range_rate,latitude,longitude,elevation]...
   =textread('range_rate_only_data.txt','%d %d %f %f %f %f %f %f',-1);
end

% Data for full state vector
if(data_type == 7)
load('satellite_position_velocity_data.txt','-ascii')
[ob_type,order_ob,JDay,ob_time,r_i,r_j,r_k,v_i,v_j,v_k,latitude,longitude,elevation]...
   =textread('satellite_position_velocity_data.txt','%d %d %f %f %f %f %f %f %f %f %f
%f %f',-1);
end

% Data for pseudo-range
if(data_type == 8)
load('satellite_position_only_data.txt','-ascii')
[ob_type,order_ob,JDay,ob_time,r_i,r_j,r_k,latitude,longitude,elevation]...
   =textread('satellite_position_only_data.txt','%d %d %f %f %f %f %f %f %f %f',-1);
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Determine the number of observations which will determine the number
% of times through the data processing loop.
num_obs = length(ob_type);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Set up z, the total data (observation) vector.
% The "order" of z is the number of "types" of data
% associated with a single observation time.  For example,
% if processing range and range-rate then the order is 2.
% The dimension of z is (number of obs) x (order).
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Observation data type 1 is for range in kilometers and
% azimuth and elevation in degrees.
if (ob_type(1) == 1)
   z = [range az el];
end

% Observation data type 2 is for range in kilometers and
% range-rate in km/s.
```

```
if (ob_type(1) == 2)
   z = [range range_rate];
end

% Observation data type 3 is for azimuth and elevation
% in degrees.
if (ob_type(1) == 3)
   z = [az el];
end

% Observation data type 4 is for topocentric right
% ascension and declination in degrees.
if (ob_type(1) == 4)
   z = [ra dec];
end

% Observation data type 5 is for range-rate only in km/s.
if (ob_type(1) == 5)
   z = [range_rate];
end

% Observation data type 7 is for the state vector.
if (ob_type(1) == 7)

   z = [r_i r_j r_k v_i v_j v_k ];

end

% Observation data type 7 is for the state vector.
if (ob_type(1) == 8)

      z = [r_i r_j r_k];

end
% Determine size of the z vector for debugging.
z_size = size(z);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Set maximum number of iterations.
max_iter = 100;

% Initialize iteration so the 'while' loop will begin processing.
iteration = 1;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%  Begin iteration loop for Non-Linear Least Squares

while iteration <= max_iter

   % "Mode" value is the flag for deciding whether only the equations
   % of motion (EOM)(mode = 0) or EOM and equations of
   % variation (EOM + EOV)(mode = 1) are processed in subroutine "rhs"
   % which provides the differential equations to be integrated.

   mode = 1;

   %%%%%%%%%%%%%%%%%%%%%%%%%%
   %
   % Initialize the "total" state vector.
   %
   %%%%%%%%%%%%%%%%%%%%%%%%%%

   % Initialize the state transition matrix, phi,

   phi = eye(6);

   % Initial "total" state conditions.  Reference trajectory and
   % the state transition matrix as a 42 x 1 column vector
   % since 'ode45' expects a column vector.
   X = [X_ref;...
         phi(1,1); phi(1,2); phi(1,3); phi(1,4); phi(1,5); phi(1,6);...
         phi(2,1); phi(2,2); phi(2,3); phi(2,4); phi(2,5); phi(2,6);...
         phi(3,1); phi(3,2); phi(3,3); phi(3,4); phi(3,5); phi(3,6);...
         phi(4,1); phi(4,2); phi(4,3); phi(4,4); phi(4,5); phi(4,6);...
         phi(5,1); phi(5,2); phi(5,3); phi(5,4); phi(5,5); phi(5,6);...
         phi(6,1); phi(6,2); phi(6,3); phi(6,4); phi(6,5); phi(6,6)];

   % Verify X is a 42 x 1 column vector for debugging.
   X_size = size(X);

   X_calc = X;

   %%%%%%%%%%%%%%%%%%%%%%%%%%
   %
   % Initialize buffers for matrix product accumulation.
   %
   % The matrices used in this program are:
```

```
%   phi - state transition matrix (6 x 6)
%   H   - observation model (order_obs x 6)
%   T   - observation matrix; product of H * phi; (order_obs x 6)
%   Q   - instrument covariance matrix (order_obs x order_obs)
%   r   - residual vector
%   P   - state covariance matrix (6 x 6)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%

% Initialize state correction term, dx, to zero first
del_X = zeros(6,1);

% For product of (T transpose)(Q inverse)(r)
% Dimensions:     (6 x n)    (n x n)   (n x 1) = (6 x 1)
T_tran_Q_inv_r = zeros(6,1);

% Initialize covariance matrix inverse (6 x 6)
P_inv = zeros(6);
% Recall P = product of (T transpose)(Q inverse)(T)

%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Observation processing loop
%
%%%%%%%%%%%%%%%%%%%%%%%%%%

for iob = 1:num_obs

   % Write iob value to screen for progress monitoring.
    fprintf('Iteration %d of %d\n',iteration,max_iter)
    fprintf('%d of %d observations is processing.\n',...
       iob,num_obs)
    fprintf('\n')
    % Blank line for spacing on screen.

   % Julian Date needed by function 'LST' for
   % determining tracking site inertial position.
   JD = JDay(iob);

   % The "order" of the observation determines the
   % number of rows in the observation z matrix.
   ndata = order_ob(iob);

   % Z matrix data type identifier to pass to function 'obser'
```

```
% so that proper section can process obs and return the
% predicted data, zpred, matrix and the H matrix.
ztype = ob_type(iob);

%%%%%%%%%%%%%%%%%%%%
%
% Numerically integrate state and state transition matrix
% derivatives to observation time.
%
%%%%%%%%%%%%%%%%%%%%

% Determine the time_vector for ode45 based on and ensure
% the dynamics are progressing forward in time

if (iob == 1)
    % time_vec = 0:ob_time(1);
    % time_vec = 0:300:(JDay(1) - JD_ref)*86400.0
    % time_vec = 0:time_step(iob); % read in with satellite r,v data
    time_step = (JDay(1) - JD_epoch)*86400.0
    time_vec = 0:time_step;
end

if(iob > 1)
    % time_step = ob_time(iob) - ob_time(iob-1);
   time_step = (JDay(iob) - JDay(iob-1))*86400.0;
   time_vec = 0:time_step;
end

check_forward_motion = time_step; % Are we going forward in time?

if check_forward_motion >= 1; %sec

abs_tol = 1e-7* ones(42,1);

options = odeset('RelTol', 1e-7, 'AbsTol', abs_tol);

% Propogate X_nom to find X_calc

% ode45 is one of MATLAB's built-in numerical integrators.  It is
% based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince
% pair.  It is a one-step solver in computing X(t), it needs only
% the solution at the immediately preceding time point, X(t n-1).

% Format of the integration routine call:
```

```
%   @rhs is the function containing the equations to be integrated.
%   time_vec is the time span to be integrated over.
%   X is the current state of the system  (initial conditions).
%   options contain the information for absolute/relative
%   tolerances, etc.

mode = 1;
[t,Y] = ode45(@ground_rhs,time_vec,X_calc,options,mode,JD,...
   third_body_flag,J2_flag, drag_flag,drag_coefficient,...
   sat_mass,sat_area,iob);

% ode45 returns a matrix that is of dimensions
% (# of times steps x # of equations integrated)
Y_ode_size = size(Y);
Y_length = length(Y);
last_row = Y_ode_size(1);

% Extract only the last time step values of state X
% because ode45 expects a 42 component column vector.
X_calc = Y(last_row,:)';

new_X_size = size(X_calc);

% Write satellite position and velocity to output file.
fprintf(fid6,'%15.5f %15.6f %15.6f %15.6f %15.6f %15.6f %15.6f\n',...
ob_time(iob),X_calc(1),X_calc(2),X_calc(3),X_calc(4),X_calc(5),X_calc(6));

long = longitude(iob);
lat = latitude(iob);
alt = elevation(iob);

%%%%%%%%%%%%%%%%%%%%
% Read Julian Date associated with this observation.
JD = JDay(iob);

% Call to subroutine 'lstime' to get Local Sidereal Time, LST
% in degrees which is used in subroutine 'obser' for
% calcualating tracking site inertial position.
[GST,LST] = lstime(JD,long);

% Call to subroutine 'obser' to get predicted data vector,
% zpred, H matrix, Q_inv matrix.
[zpred,H,Q_inv] = obser(X_calc,ztype,lat,alt,LST);
```

```
zpred_size = size(zpred);

%%%%%%%%%%%%%%%%%%%%%%
% Initialize the residual rejection flag.
rejected = 0;

if(ob_type(iob) == 1)
   r = [z(iob,1) - zpred(1);
        z(iob,2) - zpred(2);
        z(iob,3) - zpred(3)]
end


if((ob_type(iob) == 2) | (ob_type(iob) == 3) | (ob_type(iob) == 4))
   r = [z(iob,1) - zpred(1);
        z(iob,2) - zpred(2)]
end

if(ob_type(iob) == 5)
   r = [z(iob,1) - zpred(1)]
end

if(ob_type(iob) == 7)
   r = [z(iob,1) - zpred(1);
       z(iob,2) - zpred(2);
       z(iob,3) - zpred(3);
       z(iob,4) - zpred(4);
       z(iob,5) - zpred(5);
       z(iob,6) - zpred(6)];
end

if(ob_type(iob) == 8)
   r = [z(iob,1) - zpred(1);
       z(iob,2) - zpred(2);
       z(iob,3) - zpred(3)];
end

residual_vector_size = size(r);

% Write observed and predicted range, azimuth, and elevation
% and residuals to output file.
if(ob_type(iob) == 1)
   fprintf(fid1,'%15.5f %14.5f %14.8f %14.8f %14.8f %12.6f %12.6f %12.6f %11.6f
%11.6f %11.6f\n',...
```

```
      JD,ob_time(iob),z(iob,1),zpred(1),r(1),z(iob,2),zpred(2),...
      r(2),z(iob,3),zpred(3),r(3));

end


% Write observed and predicted range and range rate and
% residuals to output file.
if(ob_type(iob) == 2)
   fprintf(fid2,'%15.5f %14.5f %14.8f %14.8f %14.8f %10.6f %10.6f %10.6f\n',...
      JD,ob_time(iob),z(iob,1),zpred(1),r(1,1),z(iob,2),zpred(2),...
      r(2,1));
end


% Write observed and predicted azimuth and elevation and
% residuals to output file.
if(ob_type(iob) == 3)
   fprintf(fid3,'%15.5f %14.5f %10.6f %10.6f %10.6f %9.6f %9.6f %9.6f\n',...
      JD,ob_time(iob),z(iob,1),zpred(1),r(1),z(iob,2),zpred(2),...
      r(2));
end

% Write observed and predicted topocentric right ascension and
% declination and residuals to output file.
if(ob_type(iob) == 4)
   fprintf(fid4,'%15.5f %14.5f %10.6f %10.6f %10.6f %9.6f %9.6f %9.6f\n',...
      JD,ob_time(iob),z(iob,1),zpred(1),r(1),z(iob,2),zpred(2),...
      r(2));
end

% Write observed and predicted range rate residuals to output file.
if(ob_type(iob) == 5)
   fprintf(fid5,'%15.5f %14.5f %10.6f %10.6f %10.6f\n',...
      JD,ob_time(iob),z(iob,1),zpred(1),r(1));
end

% Write observed and state
% and residuals to output file.
if(ob_type(iob) == 7)

   rz = [z(iob,1) z(iob,2) z(iob,3) ];
   rznorm = norm(rz);
   vz = [z(iob,4) z(iob,5) z(iob,6) ];
```

```
   vznorm = norm(vz);

   rp = [zpred(1) zpred(2) zpred(3) ];
   rpnorm = norm(rp);
   vp = [zpred(4) zpred(5) zpred(6) ];
   vpnorm = norm(vp);

   xres = rznorm - rpnorm;
   vres = vznorm - vpnorm;

   fprintf(fid11,'%15.5f %14.5f %14.8f %14.8f %14.8f %12.6f %12.6f %12.6f\n',...
      JD,ob_time(iob),rznorm,rpnorm,xres,vznorm,vpnorm,vres);

end

% write observed range and residuals to file
if(ob_type(iob) == 8)

   rz = [z(iob,1) z(iob,2) z(iob,3) ];
   rznorm = norm(rz);

   rp = [zpred(1) zpred(2) zpred(3) ];
   rpnorm = norm(rp);

   xres = rznorm - rpnorm;


   fprintf(fid12,'%15.5f %14.5f %14.8f %14.8f %14.8f\n',...
      JD,ob_time(iob),rznorm,rpnorm,xres);
end


         reject = 300000.0;  % Wiesel did a number this large.

        for i = 1:ndata
            % Compare the elements of r(i) with its corresponding
            % diagonal entry of the Q_inv matrix.
            if(abs(r(i)) > reject/sqrt(Q_inv(i,i)))
               % Set residual rejection flag to sort/omit rejected obs.
               rejected = 1;
            end
        end  % End for i = 1:ndata
```

```
% If the observation is not rejected, process its matrices.
% Check if 'rejected' is anything other than 1 (not equal to 1).
   if (rejected ~= 1)

   % Extract phi matrix in normal form from the total "state"
   % column vector X_calc calculated by ground_rhs

 phi = [X_calc(7) X_calc(8) X_calc(9) X_calc(10) X_calc(11) X_calc(12);
      X_calc(13) X_calc(14) X_calc(15) X_calc(16) X_calc(17) X_calc(18);
      X_calc(19) X_calc(20) X_calc(21) X_calc(22) X_calc(23) X_calc(24);
      X_calc(25) X_calc(26) X_calc(27) X_calc(28) X_calc(29) X_calc(30);
      X_calc(31) X_calc(32) X_calc(33) X_calc(34) X_calc(35) X_calc(36);
      X_calc(37) X_calc(38) X_calc(39) X_calc(40) X_calc(41) X_calc(42)];

  % Matrix dimension statements for debugging.
   % Remove ; at end of line to write to screen.
   H_size = size(H);
   phi_size = size(phi);

   % Form matrix product  T    =    H   *  phi
   % Dimensions:       (n x 6) = (n x 6)*(6 x 6), where
   % n = ndata = order_ob
   % T is the observation matrix.
   T = H * phi;
   T_size = size(T);

   % Form product P_inv = (T transpose)*(Q inverse)*(T)
   % This product is the "observability condition."  It
   % must be invertible for an estimate to exist.
   % Dimensions:   (6 x 6)= (6 x n)*(n x n)    *(n x 6)
   P_inv = P_inv + (T' * Q_inv * T);

   % Matrix dimension statements for debugging.
   Q_inv_size = size(Q_inv);
   T_size = size(T);
   T_trans_size = size(T');
   r_size = size(r);

   % Form product (T transpose)*(Q inverse)*(r)
   % Dimensions: (6 x 1) = (6 x 1) + (6 x n)*(n x n)*(n x 1)
   T_tran_Q_inv_r = T_tran_Q_inv_r + (T' * Q_inv * r);


   end  % End to go with check of rejected ~= 1.
```

```
    rejected = 0;  % Reset rejected flag to 0 so that the next
                   % observation will be evaluated.

   end % End of loop for check_forward_motion logic check


end % End of for loop for iob = 1:num_obs


    %%%%%%%%%%%%%%%%%%%%
    %
    % Data is processed; improve estimate of xref at EPOCH.
    % You are NOT calculating xref at every time step.
    %
    %%%%%%%%%%%%%%%%%%%%

    % Invert matrix T transpose Q inverse T to find covariance P
    % Dimensions: (6 x 6) = inv((6 x n)*(n x n)*(n x 6))
    %         P = inv(T' * Q_inv * T);
      P = inv(P_inv);

    % Multiply P by T transpose Q inverse r to get correction
    % to the state vector.

    %%%%%%%%%%%%%%%%%%%%
    Vallado_Scale = (.01); % Make a correction that is 1% of
    % what is predicted by the residuals (note that Wiesel's
    % method gives a correction roughly equal to an average
    % of the residuals) (Based on Vallado finite differencing)
    %%%%%%%%%%%%%%%%%%%%

    % Dimensions: (6 x 1) = (6 x 6)*(6 x 1)

    del_X = (P * T_tran_Q_inv_r).*Vallado_Scale;

    % Establish the convergence criteria.

    this_close = .005; % .01 = 1%, .1 = 10%

    if((abs(del_X(1) > this_close*sqrt(abs(P(1,1)))))...
          | (abs(del_X(2) > this_close*sqrt(abs(P(2,2)))))...
          | (abs(del_X(3) > this_close*sqrt(abs(P(3,3)))))...
          | (abs(del_X(4) > this_close*sqrt(abs(P(4,4)))))...
```

```
            | (abs(del_X(5) > this_close*sqrt(abs(P(5,5)))))...
            | (abs(del_X(6) > this_close*sqrt(abs(P(6,6))))))

         % The vertical bar(s) in the above 'if' statement is/are
         % MATLAB's logical 'or' operator.

            convergence = 0;

         else
            convergence = 1;

      end

   del_X

   % Add in state corrections to reference state (trajectory)
   % This for the state at EPOCH only.  NOT every time step.

   X_ref = X_ref + del_X

   % Write current value of X_tgt_ref after corrections have been added.
   fprintf(fid7,'%3d %15.6f %15.6f %15.6f %15.6f %15.6f %15.6f %15.6f
%15.6f %15.6f %15.6f %15.6f\n',...
   iteration,X_ref(1),X_ref(2),X_ref(3),X_ref(4),X_ref(5),X_ref(6),...
   del_X(1),del_X(2),del_X(3),del_X(4),del_X(5),del_X(6));

   % Write the current JD as the epoch to use for future Bayes estimates
   fprintf(fid9,'%15.5f %15.6f %15.6f %15.6f %15.6f %15.6f %15.6f\n',...
      JD_epoch,X_ref(1),X_ref(2),X_ref(3),X_ref(4),X_ref(5),X_ref(6));

    if(convergence == 1)

       % Write iteration value to screen before increasing
       % and exiting loop.
       fprintf('Converged on iteration %d of %d\n',...
          iteration,max_iter);

       % Just add a number to get iterations to
       % exceed maximum and exit for loop;
       iteration = max_iter + 5;

    end

     % Increment iteration value by 1
```

```
      iteration = iteration + 1;

 end  % End statement for the iterations to max_iter loop

 % Write covariance matrix P to screen.
 P

 % Write the final covariance matrix components to output file here.
for i = 1:6
fprintf(fid8,'%25.20f %25.20f %25.20f %25.20f %25.20f %25.20f\n',...
   P(i,1),P(i,2),P(i,3),P(i,4),P(i,5),P(i,6));
end

% Principal error axes
% Extract the 3 x 3 space and velocity covariance submatrices

% Space covariance is the upper lefthand 3 x 3 of the P matrix
% Velocity covariance is the lower righthad 3 x 3 of the P matrix
for i = 1:3
   for j = 1:3
      space_P(i,j) = P(i,j);
      velocity_P(i,j) = P(i+3,j+3);
   end
end

% Eigenvector/value analysis of the covariance matrices
[V,D] = eig(space_P);
[W,E] = eig(velocity_P);

for i = 1:3
   raxis(i) = D(i,i);
   vaxis(i) = E(i,i);
end

for i = 1:3
   if(raxis(i) < 0.0)
      negative_r_axis = 'Space covariance has a negative value!'
   else
      raxis(i) = sqrt(raxis(i));
   end
end

for i = 1:3
   if(vaxis(i) < 0.0)
```

```
        negative_v_axis = 'Velocity covariance has negative value!'
    else
        vaxis(i) = sqrt(vaxis(i));
    end
end

raxis
vaxis

 % Close output files
 fclose(fid1);
 fclose(fid2);
 fclose(fid3);
 fclose(fid4);
 fclose(fid5);
 fclose(fid6);
 fclose(fid7);
 fclose(fid8);
 fclose(fid9);
 fclose(fid10);
 fclose(fid11);
 fclose(fid12);

 does_it_work = 'T-minus 15 days and counting!';

% Call plot function
% This version of 'plotted' gives the last pass residuals as it converged
[plotted] = plot_residuals(data_type,num_obs,iteration,JD_epoch)

% end of file
```

Appendix F: Subroutine Ground RHS MATLAB Code

```
function dX = ground_rhs(t,X_calc,mode,JD,third_body_flag,J2_flag,...
   drag_flag,drag_coefficient,sat_mass,sat_area,iob)


% function - calcs EOM and EOV for sat orbiting earth
% LtCol John Heslin
% 15 December 2004
% adapted from code written by Capt Brian L. Foster
% 27 January 2003

% added assignment to X since we're bring X_calc in from filter
X = X_calc;

% This MATLAB code modeled after FORTRAN code written by
% Dr. William E. Wiesel for MECH 731 Modern Methods of
% Orbit Determination.

% This function calculates the equations of motion (EOM) and/or
% not and the equations of variation (EOV) for the problem of
% a spacecraft in orbit around the Earth.

% X is the 42-component 'total' state vector
%   X(1-3) are the x,y,z components of the position vector
%   X(4-6) are the x,y,z components of the velocity vector
%   X(7-42) are the (6 x 6) state transition matrix Phi
%   stored row by row

% dX is the 42-component state vector derivatives
%   dX(1-3) are the x,y,z derivatives of position (velocity)
%   dX(4-6) are the x,y,z derivatives of velocity (acceleration)
%   dX(7-42) are the derivatives of the state transition matrix, phi dot

% Open output files for the various acceleration components
% The 'w+' instructs MATLAB that the file can be both read and written
% to and that any previous data in the file is overwritten.
fid1 = fopen('gravity_accleration_output.txt','w+');
fid2 = fopen('J2_acceleration_output.txt','w+');
fid3 = fopen('drag_acceleration_output.txt','w+');
fid4 = fopen('totatl_acceleration_output.txt','w+');


% Earth radius, RE, in kilometers
```

```
RE = 6378.1363;

% Earth gravitational parameter, mu, in km^3/sec^2
% mu_earth = 398600.4415;
mu_earth = 398600.4418;  %Vallado


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Equations of Motion.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The N-Body Problem with the origin at the center of the Earth.
% Reference Vallado pages 116-119 or Bate, Mueller, and White page 10.

% Position derivatives = velocity
dX(1) = X(4); % km/sec
dX(2) = X(5);
dX(3) = X(6);



% Velocity derivatives = gravity acceleration due to the Earth
r_vector = [X(1); X(2); X(3)];
r_norm = norm(r_vector);

f_earth(4) = - mu_earth*X(1)/r_norm^3; % basic dX(4) km/s^2
f_earth(5) = - mu_earth*X(2)/r_norm^3; % basic dX(5)
f_earth(6) = - mu_earth*X(3)/r_norm^3; % basic dX(6)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Calculate 3rd body perturbation accelerations, if desired.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Sun's gravitational parameter, km^3/s^2
mu_sun = 1.32712428e11;

% Call function 'Sun' for Sun's GEOCENTRIC position vector in km
[r_sun] = Sun(JD);
```

```
% Vector from Sun to satellite
dx_sun = X(1) - r_sun(1);
dy_sun = X(2) - r_sun(2);
dz_sun = X(3) - r_sun(3);

% Distance from the Sun to the satellite cubed
r32_sun = (dx_sun^2 + dy_sun^2 + dz_sun^2)^(3/2);

% Distance from center of Earth (central body) to Sun cubed.
rp132_sun = (r_sun(1)^2 + r_sun(2)^2 + r_sun(3)^2)^(3/2);

% Acceleration terms due to Sun; 3rd body form of the equations
f_sun(4) = -mu_sun*(dx_sun/r32_sun - r_sun(1)/rp132_sun);
f_sun(5) = -mu_sun*(dy_sun/r32_sun - r_sun(2)/rp132_sun);
f_sun(6) = -mu_sun*(dz_sun/r32_sun - r_sun(3)/rp132_sun);


% Moon's gravitational parameter, km^3/s^2
mu_moon = 4902.799;


% Call function 'Moon' for Moon's GEOCENTRIC position vector in km
[r_moon] = Moon(JD);

% Vector from Moon to the satellite
dx_moon = X(1) - r_moon(1);
dy_moon = X(2) - r_moon(2);
dz_moon = X(3) - r_moon(3);


% Distance from the Moon to the satellite cubed
r32_moon = (dx_moon^2 + dy_moon^2 + dz_moon^2)^(3/2);

% Distance from center of Earth (central body) to the Moon cubed
rp132_moon = (r_moon(1)^2 + r_moon(2)^2 + r_moon(3)^2)^(3/2);

% Acceleration terms due to the Sun; 3rd body form of equations
f_moon(4) = -mu_moon*(dx_moon/r32_moon - r_moon(1)/rp132_moon);
f_moon(5) = -mu_moon*(dy_moon/r32_moon - r_moon(2)/rp132_moon);
f_moon(6) = -mu_moon*(dz_moon/r32_moon - r_moon(3)/rp132_moon);



if(third_body_flag == 0)
```

```
   f_sun(4)   = 0.0;
   f_sun(5)   = 0.0;
   f_sun(6)   = 0.0;

   f_moon(4)  = 0.0;
   f_moon(5)  = 0.0;
   f_moon(6)  = 0.0;

end     % 'end' statement to go with third body flag check

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Calculate the perturbation of the Earth's oblateness due to J2.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% J2 gravitational zonal coefficient from JGM-2 from Appendix D
% of Vallado (1997).
J2 = -0.1082626925638815e-2;

r_i = X(1);
r_j = X(2);
r_k = X(3);

% Second harmonic J2 terms, km/s^2
f_J2(4) = -3*J2*mu_earth*(RE^2)*r_i/(2*r_norm^5)*(1-((5*r_k^2)/r_norm^2));

f_J2(5) = -3*J2*mu_earth*(RE^2)*r_j/(2*r_norm^5)*(1-((5*r_k^2)/r_norm^2));

f_J2(6) = -3*J2*mu_earth*(RE^2)*r_k/(2*r_norm^5)*(3-((5*r_k^2)/r_norm^2));



if(J2_flag == 0)

   f_J2(4) = 0.0;
   f_J2(5) = 0.0;
   f_J2(6) = 0.0;

end %end statement for J2 flag


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%
% Calculate the perturbation effect of atmospheric drag.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Earth rotational rate in rad/s.
earth_rotation_rate = 0.000072921158553;

% Calculate the satellite's velocity vector relative to the
% Earth's rotating atmosphere.

% Relative velocity, km/s.
v_rel(1) = X(4) + earth_rotation_rate * X(2);
v_rel(2) = X(5) - earth_rotation_rate * X(1);
v_rel(3) = X(6);

% Magnitude of relative velocity, km/s.
v_rel_mag = norm(v_rel);

% Determine altitude above Earth's surface, km.
altitude = r_norm - RE;

% Call function 'atmosphere' to get atmospheric density.
[density,scale_height] = atmosphere(altitude);

% % Drag acceleration terms.
% drag_coefficient -- dimensionless
% sat_area -- (m^2) * (1km/1000m)^2 = km^2
% sat_mass --(kg)
% density -- (kg/m^3) * (1000m/km)^3 = kg/km^3
% vel -- (km/s)
% gives

f_drag(4) = -0.5 * (drag_coefficient * sat_area / sat_mass)...
   * density * v_rel_mag * v_rel(1) * 1000.0;

f_drag(5) = -0.5 * (drag_coefficient * sat_area / sat_mass)...
   * density * v_rel_mag * v_rel(2) * 1000.0;

f_drag(6) = -0.5 * (drag_coefficient * sat_area / sat_mass)...
   * density * v_rel_mag * v_rel(3) * 1000.0;

% Drag acceleration terms.
```

```
if(drag_flag == 0)

   f_drag(4) = 0.0;
   f_drag(5) = 0.0;
   f_drag(6) = 0.0;

end  % 'end' statement to go with drag_flag check.

% Total acceleration for the equations of motion.
dX(4) = f_earth(4) + f_J2(4) + f_drag(4) + f_sun(4) + f_moon(4);
dX(5) = f_earth(5) + f_J2(5) + f_drag(5) + f_sun(5) + f_moon(5);
dX(6) = f_earth(6) + f_J2(6) + f_drag(6) + f_sun(6) + f_moon(6);

%debugging
dX_no_variation = [dX(1); dX(2); dX(3); dX(4); dX(5); dX(6)];

dx_no_variation2 = [iob dX(1) dX(4)];

% if(mode ~= 1)
%     dX =
% end % end statement to go with mode check
%

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% EQUATIONS OF VARIATION
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% See Vallado Fund. 2ed, pg 743
% The basic Jacobian is A:

% If mode = 1, then the equations of variation are processed.

% Calculate the A matrix (A = gradient of vector f).
% Initialize to 0 first.
A = zeros(6,6);

% A is a 6 x 6 matrix.
% The upper right 3 x 3 corner is an identity matrix.
A(1,4) = 1.0;
A(2,5) = 1.0;
A(3,6) = 1.0;
```

```
% Diagonal terms of the A matrix lower left corner 3 x 3
A(4,1) = -mu_earth/r_norm^3 + 3*mu_earth*X(1)^2/r_norm^5;
A(5,2) = -mu_earth/r_norm^3 + 3*mu_earth*X(2)^2/r_norm^5;
A(6,3) = -mu_earth/r_norm^3 + 3*mu_earth*X(3)^2/r_norm^5;

% Off-diagonal terms of the A matrix lower left corner 3 x 3
% Use symmetry to avoid as much calculation as possible.
A(4,2) = 3*mu_earth*X(1)*X(2)/r_norm^5;
A(5,1) = A(4,2);
A(4,3) = 3*mu_earth*X(1)*X(3)/r_norm^5;
A(6,1) = A(4,3);
A(5,3) = 3*mu_earth*X(2)*X(3)/r_norm^5;
A(6,2) = A(5,3);

% Equations of variation due to third body effects

% Sun's gravitational parameter, km^3/s^2
mu_sun = 1.32712428e11;

% Call function 'Sun' for Sun's GEOCENTRIC position vector in km
[r_sun] = Sun(JD);

% Vector from Sun to satellite
dx_sun = X(1) - r_sun(1);
dy_sun = X(2) - r_sun(2);
dz_sun = X(3) - r_sun(3);

% Distance from the Sun to the satellite cubed
r32_sun = (dx_sun^2 + dy_sun^2 + dz_sun^2)^(3/2);

% Distance from the Sun to the satellite to fifth power
r52_sun = r32_sun^(5/3);

% Diagonal terms for the Sun
A_sun(4,1) = -mu_sun*(1/r32_sun - 3 * dx_sun^2/r52_sun);

A_sun(5,2) = -mu_sun*(1/r32_sun - 3 * dy_sun^2/r52_sun);

A_sun(6,3) = -mu_sun*(1/r32_sun - 3 * dz_sun^2/r52_sun);



% Sun's x and y terms
```

```
A_sun(4,2) = 3*mu_sun*dx_sun*dy_sun/r52_sun;
A_sun(5,1) = A_sun(4,2);

% Sun's x and z terms
A_sun(4,3) = 3*mu_sun*dx_sun*dz_sun/r52_sun;
A_sun(6,1) = A_sun(4,3);

% Sun's y and z terms
A_sun(5,3) = 3*mu_sun*dy_sun*dz_sun/r52_sun;
A_sun(6,2) = A_sun(5,3);

% Moon's gravitational parameter, km^3/s^2
mu_moon = 4902.799;


% Call function 'Moon' for Moon's GEOCENTRIC
% position vector in km
[r_moon] = Moon(JD);

% Vector from Moon to the satellite
dx_moon = X(1) - r_moon(1);
dy_moon = X(2) - r_moon(2);
dz_moon = X(3) - r_moon(3);


% Distance from the Moon to the satellite cubed
r32_moon = (dx_moon^2 + dy_moon^2 + dz_moon^2)^(3/2);

% Distance from the Moon to the satellite to fifth power
r52_moon = r32_moon^(5/3);

% Diagonal terms for the Moon
A_moon(4,1) = -mu_moon*(1/r32_moon - 3 * dx_moon^2/r52_moon);

A_moon(5,2) = -mu_moon*(1/r32_moon - 3 * dy_moon^2/r52_moon);

A_moon(6,3) = -mu_moon*(1/r32_moon - 3 * dz_moon^2/r52_moon);


% Sun's x and y terms
A_moon(4,2) = 3*mu_moon*dx_moon*dy_moon/r52_moon;
A_moon(5,1) = A_moon(4,2);

% Sun's x and z terms
```

```
A_moon(4,3) = 3*mu_moon*dx_moon*dz_moon/r52_moon;
A_moon(6,1) = A_moon(4,3);

% Sun's y and z terms
A_moon(5,3) = 3*mu_moon*dy_moon*dz_moon/r52_moon;
A_moon(6,2) = A_moon(5,3);



if(third_body_flag == 0)

   A_sun(4,1) = 0.0;
   A_sun(5,2) = 0.0;
   A_sun(6,3) = 0.0;
   A_sun(4,2) = 0.0;
   A_sun(5,1) = 0.0;
   A_sun(4,3) = 0.0;
   A_sun(6,1) = 0.0;
   A_sun(5,3) = 0.0;
   A_sun(6,2) = 0.0;

   A_moon(4,1) = 0.0;
   A_moon(5,2) = 0.0;
   A_moon(6,3) = 0.0;
   A_moon(4,2) = 0.0;
   A_moon(5,1) = 0.0;
   A_moon(4,3) = 0.0;
   A_moon(6,1) = 0.0;
   A_moon(5,3) = 0.0;
   A_moon(6,2) = 0.0;

end

% Equations of variations due to J2
% J2 * mu? Yes

A_J2(4,1) = -(3/2)*J2*mu_earth*RE^2*((1-(5*X(3)^2)/r_norm^2)*...
   (1/r_norm^5 - (5*X(1)^2)/r_norm^7) + 10 * (X(1)^2)*(X(3)^2)/r_norm^9);

A_J2(4,2) = -(3/2)*J2*mu_earth*RE^2*X(1)*((-5*X(2)/r_norm^7)*...
   (1-5*(X(3)^2)/r_norm^2) + (10*X(2)*X(3)^2)/r_norm^9);

A_J2(4,3) = -(3/2)*J2*mu_earth*RE^2*X(1)*((-5*X(3)/r_norm^7)*...
   (1-5*(X(3)^2)/r_norm^2) + (10*X(3)/r_norm^7)*(((X(3)^2)/r_norm^2)-1));
```

```
A_J2(5,1) = -(3/2)*J2*mu_earth*RE^2*X(2)*((-5*X(1)/r_norm^7)*...
   (1-5*(X(3)^2)/r_norm^2) + (10*X(1)*X(3)^2)/r_norm^9);

A_J2(5,2) = - (3/2)*J2*mu_earth*RE^2*((1-5*(X(3)^2)/r_norm^2)*...
   (1/r_norm^5 - 5*(X(2)^2)/r_norm^7) + 10*(X(2)^2)*(X(3)^2)/r_norm^9);

A_J2(5,3) = -(3/2)*J2*mu_earth*RE^2*X(2)*((-5*X(3)/r_norm^7)*...
   (1-5*(X(3)^2)/r_norm^2) + (10*X(3)/r_norm^7)*(((X(3)^2)/r_norm^2) -1));


A_J2(6,1) = -(3/2)*J2*mu_earth*RE^2*X(3)*((-5*X(1)/r_norm^7)*...
   (3-5*(X(3)^2)/r_norm^2) + (10*X(1)*X(3)^2)/r_norm^9);

A_J2(6,2) = -(3/2)*J2*mu_earth*RE^2*X(3)*((-5*X(2)/r_norm^7)*...
   (3-5*(X(3)^2)/r_norm^2) + (10*X(2)*X(3)^2)/r_norm^9);

A_J2(6,3) = -(3/2)*J2*mu_earth*RE^2*(((3-5*X(3)^2)/r_norm^2)*...
   (1/r_norm^5 - 5*(X(3)^2)/r_norm^7) +...
      (10*(X(3)^2)/r_norm^7)*(((X(3)^2)/r_norm^2)-1));


if(J2_flag == 0)

   A_J2(4,1) = 0.0;
   A_J2(4,2) = 0.0;
   A_J2(4,3) = 0.0;
   A_J2(5,1) = 0.0;
   A_J2(5,2) = 0.0;
   A_J2(5,3) = 0.0;
   A_J2(6,1) = 0.0;
   A_J2(6,2) = 0.0;
   A_J2(6,3) = 0.0;

end


% Equations of variation due to atmospheric drag.

% Earth rotation rate, rad/s.
earth_rotation_rate = 0.000072921158553;

% Calculate the satellite's velocity vector relative to
```

```
% the Earth's rotating atmosphere.

% Relative velocity, km/s.
v_rel(1) = X(4) + earth_rotation_rate * X(2);
v_rel(2) = X(5) - earth_rotation_rate * X(1);
v_rel(3) = X(6);

% Magnitude of relative velocity, km/s.
v_rel_mag - norm(v_rel);

% Determine altitude above Earth's surface, km.
altitude = r_norm - RE;

% Call function 'atmosphere' to get atmospheric density and
% scale height.
[density, scale_height] = atmosphere(altitude);
big_H = scale_height;

% Drag constant, DC, for easy programming
% Drag acceleration terms.
% drag_coefficient -- dimensionless
% sat_area -- (m^2) * (1km/1000m)^2 = km^2
% sat_mass --(kg)
% density -- (kg/m^3) * (1000m/km)^3 = kg/km^3
% vel -- (km/s)
% gives

DC = -0.5 * drag_coefficient * density * sat_area * 1000/ sat_mass;


A_drag(4,1) = DC*v_rel(1)*(-X(1)*v_rel_mag/(big_H * r_norm) -...
   earth_rotation_rate*v_rel(2)/v_rel_mag);

A_drag(4,2) = DC*(-X(2)*v_rel_mag*v_rel(1)/(big_H * r_norm) +...
   ((earth_rotation_rate*v_rel(1)^2)/v_rel_mag) + ...
   v_rel_mag * earth_rotation_rate);

A_drag(4,3) = DC*(-X(3)*v_rel_mag*v_rel(1)/(big_H * r_norm));

A_drag(4,4) = DC*((v_rel(1)^2)/v_rel_mag + v_rel_mag);

A_drag(4,5) = DC*((v_rel(1)*v_rel(2))/v_rel_mag);

A_drag(4,6) = DC*(v_rel(1)*v_rel(3)/v_rel_mag);
```

```
A_drag(5,1) = DC*(-X(1)*v_rel_mag*v_rel(2)/(big_H * r_norm) -...
   earth_rotation_rate*(v_rel(2)^2)/v_rel_mag - ...
   earth_rotation_rate*v_rel_mag);

A_drag(5,2) = DC*v_rel(2)*(-X(2)*v_rel_mag/(big_H * r_norm) +...
   earth_rotation_rate*v_rel(1)/v_rel_mag);

A_drag(5,3) = DC*(-v_rel_mag*v_rel(2)*X(3)/(big_H * r_norm));

A_drag(5,4) = DC*(v_rel(1)*v_rel(2)/v_rel_mag);

A_drag(5,5) = DC*((v_rel(2)^2)/v_rel_mag + v_rel_mag);

A_drag(5,6) = DC*(v_rel(3)*v_rel(2)/v_rel_mag);


A_drag(6,1) = DC*v_rel(3)*(-X(1)*v_rel_mag/(big_H * r_norm) -...
   earth_rotation_rate*v_rel(2)/v_rel_mag);

A_drag(6,2) = DC*v_rel(3)*(-X(2)*v_rel_mag/(big_H * r_norm) +...
   earth_rotation_rate*v_rel(1)/v_rel_mag);

A_drag(6,3) = DC*v_rel(3)*(-X(3)*v_rel_mag/(big_H * r_norm));

A_drag(6,4) = DC*(v_rel(1)*v_rel(3)/v_rel_mag);

A_drag(6,5) = DC*(v_rel(2)*v_rel(3)/v_rel_mag);

A_drag(6,6) = DC*(v_rel(3)^2/v_rel_mag+v_rel_mag);

      % debug
A_test_drag = [A_drag(4,1) A_drag(5,2) A_drag(6,3)];
A_test_drag = [A_drag(4,2) A_drag(4,3) A_drag(5,3)];
A_test_drag = [A_drag(5,1) A_drag(6,1) A_drag(6,2)];

A_test_drag = [A_drag(4,4) A_drag(5,5) A_drag(6,6)];
A_test_drag = [A_drag(4,5) A_drag(4,6) A_drag(5,6)];
A_test_drag = [A_drag(5,4) A_drag(6,4) A_drag(6,5)];


if(drag_flag == 0)
```

```
    A_drag(4,1) = 0.0;
    A_drag(4,2) = 0.0;
    A_drag(4,3) = 0.0;
    A_drag(4,4) = 0.0;
    A_drag(4,5) = 0.0;
    A_drag(4,6) = 0.0;

    A_drag(5,1) = 0.0;
    A_drag(5,2) = 0.0;
    A_drag(5,3) = 0.0;
    A_drag(5,4) = 0.0;
    A_drag(5,5) = 0.0;
    A_drag(5,6) = 0.0;

    A_drag(6,1) = 0.0;
    A_drag(6,2) = 0.0;
    A_drag(6,3) = 0.0;
    A_drag(6,4) = 0.0;
    A_drag(6,5) = 0.0;
    A_drag(6,6) = 0.0;
end

% MAKE SURE THIS 'end' STATEMENT IS ACTIVE.  THERE IS A SOFTWARE
% BUG HERE.  IT IS NOT BLUE UNLESS YOU DELETE SPACES IN FRONT OF IT
TO
% MAKE IT ACTIVE.

% Sum the components.
% Diagonal terms.
A(4,1) = A(4,1) + A_J2(4,1) + A_drag(4,1) + A_sun(4,1) + A_moon(4,1);
A(5,2) = A(5,2) + A_J2(5,2) + A_drag(5,2) + A_sun(5,2) + A_moon(5,2);
A(6,3) = A(6,3) + A_J2(6,3) + A_drag(6,3) + A_sun(6,3) + A_moon(6,3);

% Off-diagonal terms.
A(4,2) = A(4,2) + A_J2(4,2) + A_drag(4,2) + A_sun(4,2) + A_moon(4,2);
A(5,1) = A(5,1) + A_J2(5,1) + A_drag(5,1) + A_sun(5,1) + A_moon(5,1);

A(4,3) = A(4,3) + A_J2(4,3) + A_drag(4,3) + A_sun(4,3) + A_moon(4,3);
A(6,1) = A(6,1) + A_J2(6,1) + A_drag(6,1) + A_sun(6,1) + A_moon(6,1);

A(5,3) = A(5,3) + A_J2(5,3) + A_drag(5,3) + A_sun(5,3) + A_moon(5,3);
A(6,2) = A(6,2) + A_J2(6,2) + A_drag(6,2) + A_sun(6,2) + A_moon(6,2);
```

```
% Equations of variation that are velocity related.
A(4,4) = A_drag(4,4);
A(4,5) = A_drag(4,5);
A(4,6) = A_drag(4,6);

A(5,4) = A_drag(5,4);
A(5,5) = A_drag(5,5);
A(5,6) = A_drag(5,6);

A(6,4) = A_drag(6,4);
A(6,5) = A_drag(6,5);
A(6,6) = A_drag(6,6);

% Extract phi matrix in normal form from the total state
% column vector X.
phi = [X(7) X(8) X(9) X(10) X(11) X(12);
   X(13) X(14) X(15) X(16) X(17) X(18);
   X(19) X(20) X(21) X(22) X(23) X(24);
   X(25) X(26) X(27) X(28) X(29) X(30);
   X(31) X(32) X(33) X(34) X(35) X(36);
   X(37) X(38) X(39) X(40) X(41) X(42)];


% Calculate the derivative of the state transition matrix, phi dot.
phi_dot = A * phi;

% Write the total state derivative as a column vector to return.
dX = [dX(1); dX(2); dX(3); dX(4); dX(5); dX(6);...
   phi_dot(1,1); phi_dot(1,2); phi_dot(1,3); phi_dot(1,4);...
   phi_dot(1,5); phi_dot(1,6); phi_dot(2,1); phi_dot(2,2);...
   phi_dot(2,3); phi_dot(2,4); phi_dot(2,5); phi_dot(2,6);...
   phi_dot(3,1); phi_dot(3,2); phi_dot(3,3); phi_dot(3,4);...
   phi_dot(3,5); phi_dot(3,6); phi_dot(4,1); phi_dot(4,2);...
   phi_dot(4,3); phi_dot(4,4); phi_dot(4,5); phi_dot(4,6);...
   phi_dot(5,1); phi_dot(5,2); phi_dot(5,3); phi_dot(5,4);...
   phi_dot(5,5); phi_dot(5,6); phi_dot(6,1); phi_dot(6,2);...
   phi_dot(6,3); phi_dot(6,4); phi_dot(6,5); phi_dot(6,6)];

if(mode ~= 1)
   dX = dX_no_variation;
end % end of mode check logic

dX_size = size(dX);
```

```
dummy_dx =  [dX(1); dX(2); dX(3); dX(4); dX(5); dX(6)]; % for debugging

% Close output data files.
fclose(fid1);
fclose(fid2);
fclose(fid3);
fclose(fid4);
```

Appendix G: Subroutine Obser MATLAB Code

```
function [zpred,H,Q_inv] = obser(X_calc,ztype,lat,alt,LST)

% Capt Brian L. Foster
% 20 December 2002
% Edited by LtCol John Heslin
% 15 December 2004

% This MATLAB code modeled after FORTRAN code written by
% Dr. William E. Wiesel for MECH 731 Modern Methods of
% Orbit Determination.

% This subroutine performs the observation relation processing.
% It calculates the predicted observation, z_pred; H matrix; and
% returns the inverse of the data (instrument or measurements)
% covariance matrix, Q_inv.

format long g

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Data types:
%   1 = range, azimuth, and elevation
%   2 = range and range-rate only
%   3 = azimuth and elevation only
%   4 = topocentric right ascension and declination
%   5 = range-rate only
%   7 = full state vector
%   8 = pseudo-range



%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Inertial (ECI coordinate system) position of the tracking site.
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Earth radius in km
earth_rad = 6378.1363;

% Earth rotation rate, radians/sec
% (3 x 1 vector in the ECI coordinate system)
earth_rotation_rate = [0; 0; 0.000072921158553];
```

```
% Earth's shape eccentricity (squared) (no units)
earth_ecc_2 = 0.006694385;

% Convert input angles in degrees to radians
lat = lat * pi/180.0;
LST = LST * pi/180.0;

% Earth shape auxiliary terms:
% C_earth (a.k.a. radius of curvature in the meridian), km
C_earth = earth_rad/sqrt(1.0 - earth_ecc_2 * (sin(lat))^2);

% S_earth
S_earth = C_earth * (1.0 - earth_ecc_2);

% Horizontal component, km
r_del = (C_earth + alt) * cos(lat);

% Vertical component, km
r_K = (S_earth + alt) * sin(lat);

% Tracking site ECI position vector (3 x 1), km
r_site_ijk = [r_del * cos(LST);
   r_del * sin(LST);
   r_K];

% Tracking site ECI velocity vector (3 x 1), km/s
v_site_ijk = cross(earth_rotation_rate,r_site_ijk);

% Relative position vector (3 x 1) (range) in IJK coordinates
% from the tracking site to the satellite.  X(1) through X(3)
% represent the ECI position x,y,z coordinates from the
% current state.
range_ijk = [X_calc(1) - r_site_ijk(1);
   X_calc(2) - r_site_ijk(2);
   X_calc(3) - r_site_ijk(3)];

% Magnitude of range in IJK coordinates
range_ijk_mag = norm(range_ijk);

% Combined transformation matrix from IJK to SEZ is the
% transpose of SEZ_TO_IJK matrix.
IJK_TO_SEZ = [sin(lat)*cos(LST) sin(lat)*sin(LST) -cos(lat);
   -sin(LST) cos(LST) 0;
```

```
   cos(lat)*cos(LST) cos(lat)*sin(LST) sin(lat)];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Rotate the predicted range vector from ECI to SEZ, kilometers
range_sez = IJK_TO_SEZ * range_ijk;

% Magnitude of predicted range in kilometers.
range_sez_mag = norm(range_sez);

% Satellite inertial velocity in km/s.
v_ijk = [X_calc(4); X_calc(5); X_calc(6)];

% Relative velocity in IJK coordinates, in km/s
rel_vel_ijk = v_ijk - v_site_ijk;

% Relative velocity in SEZ coordinates, in km/s
rel_vel_sez = IJK_TO_SEZ * rel_vel_ijk;

% Magnitude of range rate in SEZ, in km/s
range_rate_mag = dot(range_sez,rel_vel_sez)/range_sez_mag;


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Range, azimuth, and elevation processing
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if(ztype == 1)

% Range calculated above outside of the "if condition."

% Predicted azimuth angle in the SEZ coordinate system, in degrees.
%az_sez = (atan(-range_sez(2)/range_sez(1)))*180.0/pi + 180.0;

% Intermediate variable since it occurs several times
rhos2rhoe2 = range_sez(1)^2 + range_sez(2)^2;
% Azimuth
   sin_az = range_sez(2)/sqrt(rhos2rhoe2);

   cos_az = -range_sez(1)/sqrt(rhos2rhoe2);

% Azimuth, in degrees
az_sez = atan2(sin_az,cos_az) * 180.0/pi;
```

```
if(az_sez < 0.0)
   az_sez = az_sez + 360.0;
end

if(az_sez > 360.0)
   az_sez = az_sez - 360.0;
end

% Predicted elevation angle in the SEZ coordinate system, in degrees.
el_sez = atan(range_sez(3)/sqrt((range_sez(2))^2 +...
   (range_sez(1))^2))*180.0/pi;

% Form predicted data vector, for this data set it is a 3 x 1
zpred = [range_sez_mag; az_sez; el_sez];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Form Q inverse matrix based on instrumentation sigmas
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Q(1,1) = 0.2^2;   % Units are km^2
% Q(2,2) = 0.003^2; %(1.432*10^-3)^2; % Units are degrees^2
% Q(3,3) = 0.003^2;%(1.432*10^-3)^2; % Units are degrees^2

% debugging:
Q(1,1) = .2^2;   % Units are km^2
Q(2,2) = .02^2/100; %(1.432*10^-3)^2; % Units are degrees^2
Q(3,3) = .02^2/100; %(1.432*10^-3)^2; % Units are degrees^2

Q_inv = inv(Q);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Form H, the observation matrix.

% Whereas G is the observation relationship, relating the data to the state
% (relating range/az/el to r/v), H is the partial of G with respect to the
% state: H = dG/dX.  For range/az,el data, Wiesel provides H as eq 3.60,
% page83

% H matrix found on pages 75-76 of Wiesel and signs changed on
% row 2 in accordance with text on page 80 to account for the
```

```
% azimuth difference.
% Initialize H to zeros first then build up needed components.

H = zeros(3,6);

% H is a 3 x 6 matrix based on SEZ coordinates.

H(1,1) = range_sez(1)/range_sez_mag;
H(1,2) = range_sez(2)/range_sez_mag;
H(1,3) = range_sez(3)/range_sez_mag;

denominator_2 = 1.0 + (range_sez(2)/range_sez(1))^2;
H(2,1) = -(range_sez(2)/(range_sez(1))^2)/denominator_2;
H(2,2) = (1.0/range_sez(1))/denominator_2;
H(2,3) = 0;

x2y2 = range_sez(1)^2 + range_sez(2)^2;
denominator_3 = 1.0 + (range_sez(3)^2)/(range_sez(1)^2 + range_sez(2)^2);
H(3,1) = -((range_sez(1)*range_sez(3))/(sqrt(x2y2)^3))/denominator_3;
H(3,2) = -((range_sez(2)*range_sez(3))/(sqrt(x2y2)^3))/denominator_3;
H(3,3) = (1.0/sqrt(x2y2))/denominator_3;

return
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Range and range-rate processing
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if(ztype == 2)

   % Range and range-rate calculated above outside of "if condition."

   % Form z, predicted data vector. (2 x 1)
   % Each component of zpred is a scalar.
   zpred = [range_sez_mag; range_rate_mag];

 % Form Q, the instrumental covariance matrix here.
 Q = zeros(2,2);

 Q(1,1) = 0.005^2;
 Q(2,2) = 0.005^2;
```

```
 Q_inv = inv(Q);

% Form H, the observation matrix, here.
% H matrix found on pages 75-76 of Wiesel and signs changed on
% row 2 in accordance with text on page 80 to account for the
% azimuth difference.
% H is a 2 x 6 matrix based on SEZ coordinates.
% Initialize H to zeros first then build up needed components.
H = zeros(2,6);

% Equations for range partial derivatives that change wrt position
denominator_1 = range_sez_mag;
H(1,1) = range_sez(1)/denominator_1;
H(1,2) = range_sez(2)/denominator_1;
H(1,3) = range_sez(3)/denominator_1;

% Equations for range-rate partial derivatives that change wrt
% position and velocity.
H(2,1) = rel_vel_sez(1)/range_sez_mag -
range_rate_mag*range_sez(1)/range_sez_mag^2;
H(2,2) = rel_vel_sez(2)/range_sez_mag -
range_rate_mag*range_sez(2)/range_sez_mag^2;
H(2,3) = rel_vel_sez(3)/range_sez_mag -
range_rate_mag*range_sez(3)/range_sez_mag^2;
H(2,4) = range_sez(1)/range_sez_mag;
H(2,5) = range_sez(2)/range_sez_mag;
H(2,6) = range_sez(3)/range_sez_mag;

return
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Azimuth and elevation (angles) only processing
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if(ztype == 3)

% Intermediate variable since it occurs several times
rhos2rhoe2 = range_sez(1)^2 + range_sez(2)^2;
% Azimuth
   sin_az = range_sez(2)/sqrt(rhos2rhoe2);
```

```
   cos_az = -range_sez(1)/sqrt(rhos2rhoe2);

% Azimuth, in degrees
az_sez = atan2(sin_az,cos_az) * 180.0/pi;

if(az_sez < 0.0)
   az_sez = az_sez + 360.0;
end

if(az_sez > 360.0)
   az_sez = az_sez - 360.0;
end

% Predicted azimuth angle in the SEZ coordinate system, in degrees.
%az_sez = (atan(-range_sez(2)/range_sez(1)))*180.0/pi;

% Predicted elevation angle in the SEZ coordinate system, in degrees.
el_sez = atan(range_sez(3)/sqrt((range_sez(2))^2 +...
   (range_sez(1))^2))*180.0/pi;

% Form predicted data vector, for this data set it is a 2 x 1
zpred = [az_sez; el_sez];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Form Q inverse matrix based on instrumentation sigmas
% Sigmas based on GBR-X radar data from literature review. (Sort of)
Q = zeros(2,2);

Q(1,1) = 0.002^2; % Units are degrees^2
Q(2,2) = 0.002^2 ;  %(1.432*10^-3)^2; % Units are degrees^2

Q_inv = inv(Q);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Form H, the observation matrix, here.
% H matrix found on pages 75-76 of Wiesel and signs changed on
% row 2 in accordance with text on page 80 to account for the
% azimuth difference.
% H is a 2 x 6 matrix based on SEZ coordinates.
% Initialize H to zeros first then build up needed components.
H = zeros(2,6);

denominator_2 = 1.0 + (range_sez(2)/range_sez(1))^2;
```

```
H(1,1) = -(range_sez(2)/(range_sez(1))^2)/denominator_2;
H(1,2) = (1.0/range_sez(1))/denominator_2;
H(1,3) = 0;

x2y2 = range_sez(1)^2 + range_sez(2)^2;
denominator_3 = 1.0 + (range_sez(3)^2)/(range_sez(1)^2 + range_sez(2)^2);
H(2,1) = -((range_sez(1)*range_sez(3))/(sqrt(x2y2)^3))/denominator_3;
H(2,2) = -((range_sez(2)*range_sez(3))/(sqrt(x2y2)^3))/denominator_3;
H(2,3) = (1.0/sqrt(x2y2))/denominator_3;


return
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Topocentric right ascension and declination processing
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if(ztype == 4)

% This is Algorithm 14 from Vallado (1997) pages 168-169.

% Topocentric declination, in degrees
dec_topo = asin(range_ijk(3)/range_ijk_mag) * 180.0/pi;

% Intermediate variable since it occurs several times
rhoi2rhoj2 = range_ijk(1)^2 + range_ijk(2)^2;

% Trigonometric quantities of topocentric right ascension
   sin_ra_topo = range_ijk(2)/sqrt(rhoi2rhoj2);

   cos_ra_topo = range_ijk(1)/sqrt(rhoi2rhoj2);

   % Topocentric right ascension, in degrees
ra_topo = atan2(sin_ra_topo,cos_ra_topo) * 180.0/pi;

if(ra_topo < 0.0)
   ra_topo = ra_topo + 360.0;
end

if(ra_topo > 360.0)
   ra_topo = ra_topo - 360.0;
```

```
end

% Form predicted data vector, for this data set it is a 2 x 1
zpred = [ra_topo; dec_topo];

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Form Q inverse matrix based on instrumentation sigmas
% Sigmas based on GBR-X radar data from literature review. (Sort of)
Q = zeros(2,2);

Q(1,1) = 0.03;
Q(2,2) = 0.03;

Q_inv = inv(Q);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Form H, the observation matrix, here.
% H matrix found on pages 75-76 of Wiesel and signs changed on
% row 2 in accordance with text on page 80 to account for the
% azimuth difference.
% H is a 2 x 6 matrix based on SEZ coordinates.
% Initialize H to zeros first then build up needed components.
H = zeros(2,6);

denominator_2 = 1.0 + (range_ijk(2)/range_ijk(1))^2;
H(1,1) = -(range_ijk(2)/(range_ijk(1))^2)/denominator_2;
H(1,2) = (1.0/range_ijk(1))/denominator_2;
H(1,3) = 0;

x2y2 = range_ijk(1)^2 + range_ijk(2)^2;
denominator_3 = 1.0 + (range_ijk(3)^2)/(range_ijk(1)^2 + range_ijk(2)^2);
H(2,1) = -((range_ijk(1)*range_ijk(3))/(sqrt(x2y2)^3))/denominator_3;
H(2,2) = -((range_ijk(2)*range_ijk(3))/(sqrt(x2y2)^3))/denominator_3;
H(2,3) = (1.0/sqrt(x2y2))/denominator_3;

return
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Range rate only processing
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
if(ztype == 5)

   % Range-rate calculated above outside of "if condition."

   % Form z, predicted data vector here.
   zpred = [range_rate_mag];

   % Equations for range-rate partial derivatives that change wrt
   % position and velocity.
   H(1,1) = rel_vel_sez(1)/range_sez_mag -...
      range_rate_mag*range_sez(1)/range_sez_mag^2;

   H(1,2) = rel_vel_sez(2)/range_sez_mag -...
      range_rate_mag*range_sez(2)/range_sez_mag^2;

   H(1,3) = rel_vel_sez(3)/range_sez_mag -...
      range_rate_mag*range_sez(3)/range_sez_mag^2;

   H(1,4) = range_sez(1)/range_sez_mag;

   H(1,5) = range_sez(2)/range_sez_mag;

   H(1,6) = range_sez(3)/range_sez_mag;

   % Instrument covariance matrix. (1 x 1)
   Q(1,1) =  0.1;
   Q_inv = inv(Q);

return
end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Range, azimuth, elevation, and range-rate processing
% and maybe azimuth-rate and elevation-rate
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if(ztype == 6)

   % Range calculated above outside of the "if condition."

% Predicted azimuth angle in the SEZ coordinate system, in degrees.
az_sez = (pi - atan(range_sez(2)/range_sez(1)))*180.0/pi;
```

```
% Predicted elevation angle in the SEZ coordinate system, in degrees.
el_sez = atan(range_sez(3)/sqrt((range_sez(2))^2 + (range_sez(1))^2))*180.0/pi;

   % This is Algorithm 15 from Vallado (1997) pages 173-174.

% Elevation, in degrees
el = asin(range_sez(3)/range_sez_mag)* 180.0/pi;

% Intermediate variable since it occurs several times
rhos2rhoe2 = range_sez(1)^2 + range_sez(2)^2;

% Azimuth
if(el ~= 90.0)
   sin_az = range_sez(2)/sqrt(rhos2rhoe2);

   cos_az = -range_sez(1)/sqrt(rhos2rhoe2);
end

if(el == 90.0)
   sin_az = range_rate_sez(2)/sqrt(range_rate_sez(1)^2 ...
      + range_rate_sez(2)^2);

   cos_az = -range_rate_sez(1)/sqrt(range_rate_sez(1)^2 ...
      + range_rate_sez(2)^2);
end

% Azimuth, in degrees
az = atan2(sin_az,cos_az) * 180.0/pi;

% Range rate magnitude
% Vallado page 174 is unclear whether the range/range rate
% used next is IJK or SEZ.  I assume SEZ.  Units in km/s.
range_rate = dot(range_sez,range_rate_sez)/range_sez_mag;

% Azimuth rate, in degrees/sec
az_rate = ((range_rate_sez(1)*range_sez(2) - ...
   range_rate_sez(2)*range_sez(1))/(rhos2rhoe2))*180.0/pi;

% Elevation rate, in degrees/sec
el_rate = ((range_rate_sez(3) - ...
   range_rate*sin(el*pi/180.0))/sqrt(rhos2rhoe2))*180.0/pi;
```

```
% Form predicted data vector (6 x 1)
zpred = [range_sez;
   az;
   el;
   range_rate_sez;
   az_rate;
   el_rate];

% Q matrix (6 x 6)
% Q = zeros(6,6);
Q = eye(6)

%Q(1,1) =
%Q(2,2) =
%Q(3,3) =
%Q(4,4) =
%Q(5,5) =
%Q(6,6) =

Q_inv = inv(Q);

% H matrix (6 x 6)
H = zeros(6,6);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Form H, the observation matrix, here.
% H matrix found on pages 75-76 of Wiesel and signs changed on
% row 2 in accordance with text on page 80 to account for the
% azimuth difference.
% Initialize H to zeros first then build up needed components.
H = zeros(3,6);

% H is a 3 x 6 matrix based on SEZ coordinates.

denominator_1 = range_sez_mag;
H(1,1) = range_sez(1)/denominator_1;
H(1,2) = range_sez(2)/denominator_1;
H(1,3) = range_sez(3)/denominator_1;

denominator_2 = 1.0 + (range_sez(2)/range_sez(1))^2;
H(2,1) = (range_sez(2)/(range_sez(1))^2)/denominator_2;
H(2,2) = -(1.0/range_sez(1))/denominator_2;
H(2,3) = 0;
```

```
x2y2 = range_sez(1)^2 + range_sez(2)^2;
denominator_3 = 1.0 + (range_sez(3)^2)/(range_sez(1)^2 + range_sez(2)^2);
H(3,1) = -((range_sez(1)*range_sez(3))/(sqrt(x2y2)^3))/denominator_3;
H(3,2) = -((range_sez(2)*range_sez(3))/(sqrt(x2y2)^3))/denominator_3;
H(3,3) = (1.0/sqrt(x2y2))/denominator_3;

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% r and v (For debugging filter performance)
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if(ztype == 7)

   % Range-rate calculated above outside of "if condition."

   % Form z, predicted data vector here.

   rangex = X_calc(1);
   rangey = X_calc(2);
   rangez = X_calc(3);
   velx = X_calc(4);
   vely = X_calc(5);
   velz = X_calc(6);


zpred = [rangex;
   rangey;
   rangez;
   velx;
   vely;
   velz];

   % Equations for range-rate partial derivatives that change wrt
   % position and velocity.

H = eye(6);

   % Instrument covariance matrix. (1 x 1)

Q1 = eye(3)*1;
Q4 = eye(3)*.01 / 1000;
```

```
Q = [Q1, zeros(3,3);zeros(3,3), Q4];
Q_inv = inv(Q);

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% pseudo-state range position only
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if(ztype == 8)

% Form z, predicted data vector here.

rangex = range_sez(1);
rangey = range_sez(2);
rangez = range_sez(3);

zpred = [rangex;
         rangey;
         rangez];

% Observation relation G = zpred
% Linearized Observation matrix H = partial(G) wrt state X
% therefore H(3x6) = [eye(3), zeros(3,3)]
H = [eye(3),zeros(3,3)];

% Instrument covariance matrix. (1 x 1)

% Intermediate variable since it occurs several times
rhos2rhoe2 = range_sez(1)^2 + range_sez(2)^2;

% Predicted Azimuth, SEZ, in degrees
   sin_az = range_sez(2)/sqrt(rhos2rhoe2);

   cos_az = -range_sez(1)/sqrt(rhos2rhoe2);

   az_sez = atan2(sin_az,cos_az) * 180.0/pi; %atan2 is MATLAB 4-quad arctan call
      if(az_sez < 0.0)
         az_sez = az_sez + 360.0;
      end

      if(az_sez > 360.0)
```

```
        az_sez = az_sez - 360.0;
      end

% Predicted Elevation, SEZ, in degrees.
   el_sez = atan(range_sez(3)/sqrt((range_sez(2))^2 +...
      (range_sez(1))^2))*180.0/pi;

%initialize Q_not
Q_not = eye(3);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
   Q_not(1,1) = .784^2; %0.784km error sigma along track
   Q_not(2,2) = .784^2; %0.784km error sigma along track
   Q_not(3,3) = .035^2; %0.035km error sigma above/below track (pancake ellisoid)
%   Q_not(3,3) = .735^2; %0.035km error sigma above/below track (spheroid)

   % Rotation matrix D_inv = inv[SEZ to IJK] = [IJK to SEZ]
   D_inv = inv(IJK_TO_SEZ); % from approx line 86 above

   % Matrix K is the partial of zpred/[range,az,el]
   rho = norm(zpred);
   K = [ -cos(el_sez)*cos(az_sez), rho*cos(el_sez)*sin(az_sez),...
      rho*sin(el_sez)*cos(az_sez);
      cos(el_sez)*sin(az_sez), rho*cos(el_sez)*cos(az_sez),...
      -rho*sin(el_sez)*sin(az_sez);
      sin(el_sez),    0,  rho*cos(el_sez)];

   % Jacobian matrix J = D_inv*K
   J = D_inv*K;

   Q = J*Q_not*J';

Q_inv = inv(Q);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Q_inv = inv(Q_not);

end
return
% end of file
```

Appendix H: Data Generator routine MATLAB Code

This is a typical data generator routine.

```
% Data generator for China Don 2N Radar Site
% Lt Col John Heslin
% Adapted from code written by Brian Foster
format long g

% Mode is a switch/flag to send to subroutine
% 'rhs' that instructs 'rhs' not to process the
% equations of variation.
mode = 0;  % Mode will ALWAYS be 0 from this subroutine.

%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Set flags to determine what kind of forces and/or
%  perturbations will be included in the propagation of the
%  orbit.  0 = perturbation off; 1 = perturbation on
%
%%%%%%%%%%%%%%%%%%%%%%%%

% Third-body gravity perturbations flag
third_body_flag = 1;

% J2 zonal gravity harmonic perturbation flag
J2_flag = 1;

% Drag force flag
drag_flag = 1;

%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Satellite properties for the drag calculation
%
%%%%%%%%%%%%%%%%%%%%%%%%

% Satellite mass in kilograms
% Shuttle mass is 171,000 lbs
sat_mass = 77565;

% Satellite cross-sectional area, in square meters
% Shuttle Drag Area : 2750.0 SQ FT = 255.48336 m^2
```

```
sat_area = 255.48336;

% Drag coefficient (dimensionless)
drag_coefficient = 1.0; % From STS-61 data on http://spacelink.nasa.gov/.index.html

%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Put initial orbit estimate (position, r, and velocity, v, here.
%  xref 1-3 are position in kilometers.
%  xref 4-6 are velocity in kilometers/second.
%  JD is the Julian Date of the epoch of the initial orbit.
%
%%%%%%%%%%%%%%%%%%%%%%%%%

% SHUTTLE STS-109
JD = 2452340.472917; %   < == grabbed from SOAP

JD_epoch = JD

xref(1) =  -2109.3158;
xref(2) = -5785.9666;
xref(3) = 3119.6134;
xref(4) = 6.82539645;
xref(5) = -3.15651721;
xref(6) = -1.12407815;

% Initialize the six components of the "state" vector, X, as a
% column vector since the integration subroutine 'ode45' will
% expect a column vector.
X = [xref(1); xref(2); xref(3); xref(4); xref(5); xref(6)];

% Verify X is a 6 x 1 column vector.
X_size = size(X);

% Orbit period in seconds is needed to establish integration
% step-size, currently set for 1,000 points for one orbit.
% period = 6113.7532549237;

%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Put tracking site data here.
%  Tracking site data for Xichang SLC, China
%
%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
lat = 28.2;         % degrees
long = 102.00;      % degrees
elevation = 2.347;  % kilometers above reference geoid

%%%%%%%%%%%%%%%%%%%%
%
%  This program was intended to process the following types of
%  data (observations).  Currently, only range and range-rate
%  work correctly.
%  ob_order is needed for dimensioning matrices/vectors
%
%%%%%%%%%%%%%%%%%%%%
%
%  ob_type:
%   1 = range, az, el (order 3)
%   2 = range and range-rate  (order 2)
%   3 = azimuth and elevation (order 2)
%   4 = topocentric right ascension and declination (order 2)
%   5 = range-rate only  (order 1)
%
%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%
%
% Open output data files
% The 'a' tells MATLAB to append previous data in the file....
% ...so Ensure Old copies of the files are Deleted first!
%
%%%%%%%%%%%%%%%%%%%%

fid  = fopen('range_az_el_data.txt','a');
fid2 = fopen('range_and_range_rate_data.txt','a');
fid3 = fopen('azimuth_and_elevation_data.txt','a');
fid4 = fopen('rt_asc_and_declination_data.txt','a');
fid5 = fopen('range_rate_only_data.txt','a');
fid6 = fopen('tracking_site_position_data.txt','a');
fid7 = fopen('satellite_position_velocity_data.txt','a');
fid8 = fopen('satellite_position_only_data.txt','a');
fid9 = fopen('satellite_state_truth_data.txt','a');

% Since the subroutine 'razel' generates range, range-rate,
% azimuth, azimuth rate, elevation, and elevation rate
% simultaneously, these data will be written to the
```

```
% appropriate file regardless of ob_type selected.

%%%%%%%%%%%%%%%%%%%%
%
%  Initialize data for MATLAB's built-in 'ode45' numerical
%  integrator function.
%
%%%%%%%%%%%%%%%%%%%%

% Time "vector" to pass to integration routine ode45.
% Currently set at 60 second intervalsl.  Change to shorter interval
% and increase the number of points to get more points more
% frequently throughout the orbit(s).

time_step = 1.1;  % seconds (radar reading every second)
% Use 1.1 seconds to ensure the ODE45 time vector is at least 1 second

time_vec = 0:time_step;

% Specify number of data points desired.
num_points = 200; % radar tracking for 3.5 minutes

% Initialize "elapsed" time for first data point.  Julian Date will
% also be included in data file.
time = time_step;

%%%%%%%%%%%%%%%%%%%%
%
%  BEGIN DATA GENERATION.
%
%%%%%%%%%%%%%%%%%%%%
%
% Write Epoch data at the start of the file
% Call to subroutine 'lstime' to get Local Sidereal Time
% for the tracking site.  (Greenwich Sidereal Time, GST,
% not used).  LST in degrees.
[GST,LST] = lstime(JD,long);
[r_site_ijk] = site(lat,long,elevation,LST);

% Earth rotation rate vector (rad/sec)
earth_rotation_rate = [0; 0; 0.000072921158553];

% Tracking site ECI velocity vector (3 x 1), km/s
v_site_ijk = cross(earth_rotation_rate,r_site_ijk);
```

```
rijk = [X(1); X(2); X(3)];
vijk = [X(4); X(5); X(6)]; % Omitted comma before 'X(5)' term (after semi-colon)

% Call to subroutine 'razel' to generate data
[range,az,el,range_rate] ...
   = razel(rijk,vijk,elevation,lat,LST);

% Call to subroutine 'topocentric' to generate
% right ascension and declination data.
[range_topo,ra_topo,dec_topo,range_rate_topo,ra_topo_rate,dec_topo_rate]...
   = topocentric(rijk,vijk,r_site_ijk);
% call to 'razel_ro' to generate psuedo-position state vector
[range_sez,az_dummy,el_dummy,range_rate_dummy]...
   = razel_ro(rijk,vijk,elevation,lat,LST);

fprintf(fid,'%1d %1d %24.16f %14.5f %12.5f %10.5f %10.5f %9.5f %10.5f %9.5f\n'...
   ,1,3,JD,time,range,az,el,lat,long,elevation);

fprintf(fid2,'%1d %1d %24.16f %14.5f %14.5f %9.5f %9.5f %10.5f %9.5f\n'...
   ,2,2,JD,time,range,range_rate,lat,long,elevation);

fprintf(fid3,'%1d %1d %24.16f %14.5f %10.5f %10.5f %9.5f %10.5f %9.5f\n'...
   ,3,2,JD,time,az,el,lat,long,elevation);

fprintf(fid4,'%1d %1d %24.16f %14.5f %10.5f %10.5f %9.5f %10.5f %9.5f\n'...
   ,4,2,JD,time,ra_topo,dec_topo,lat,long,elevation);

fprintf(fid5,'%1d %1d %24.16f %14.5f %8.5f %9.5f %10.5f %9.5f\n'...
   ,5,1,JD,time,range_rate,lat,long,elevation);

         % Write satellite position and velocity to output file
fprintf(fid7,'%1d %1d %18.8f %14.5f %14.5f %14.5f %14.5f %14.5f %14.5f %14.5f
%9.5f %10.5f %9.5f\n'...
   ,7,6,JD,time,X(1),X(2),X(3),X(4),X(5),X(6),lat,long,elevation);

   % Write satellite position only to output file
fprintf(fid8,'%1d %1d %18.8f %14.5f %14.5f %14.5f %14.5f %9.5f %10.5f %9.5f\n'...
   ,8,3,JD,time,range_sez(1),range_sez(2),range_sez(3),lat,long,elevation);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%  Now that the epoch reference is written, write data for each observation

for i = 1:num_points
```

```
% Absolute error tolerance "vector" for ode45.  This is specified
% as a 6 x 1 column vector corresponding to the size of the state
% to be integrated.

% VERY IMPORTANT: The tolerance must be set very tight (i.e. don't
% accept the MATLAB default value of 1e-4) or the orbit(s) will
% diverge quickly.  Dr. Tragesser recommends at least 1e-8 or
% smaller such as 1e-10.

abs_tol = 1e-8 * ones(6,1);

options = odeset('RelTol', 1e-8, 'AbsTol', abs_tol);


% ode45 is one of MATLAB's built-in numerical integrators.  It is
% based on an explicit Runge-Kutta (4,5) formula, the Dormand-Prince
% pair.  It is a one-step solver in computing X(t), it needs only
% the solution at the immediately preceding time point, X(t n-1).

% Format of the integration routine call:
%   @rhs is the function containing the equations to be integrated.
%   time_vec is the time span to be integrated over.
%   X is the current state of the system  (initial conditions).
%   options contain the informatiion for absolute/relative
%   tolerances, etc.

% Numerically integrate state X to output state Y.
[t,Y] = ode45(@rhs, time_vec, X, options,JD,mode,third_body_flag,...
   J2_flag, drag_flag,drag_coefficient,sat_mass,sat_area);

t;

% ode45 returns a matrix that is of dimensions
% (# of times steps x # of equations integrated).
% Since there is a large timespan and stepsize sent to
% ode45, it will return a complete data matrix with one call
% here (i.e. iteration loop is NOT required here.)

% Determine matrix dimensions of state Y, if needed.
Y_ode_size = size(Y);

% Write state Y to screen, if desired (remove ; at end of line).
Y;
```

```
% Determine length (number of rows) created from ode45
last_row_Y = Y_ode_size(1);

% Update the state X with the last row of output state Y
X = Y(last_row_Y,:);

% Call to subroutine 'lstime' to get Local Sidereal Time
% for the tracking site.  (Greenwich Sidereal Time, GST,
% not used).  LST in degrees.
[GST,LST] = lstime(JD,long);

% Call to subroutine 'site' to get instantaneous
% tracking site position vector in ECI coordinates.
% Units are in kilometers.
[r_site_ijk] = site(lat,long,elevation,LST);

% Earth rotation rate vector (rad/sec)
earth_rotation_rate = [0; 0; 0.000072921158553];

% Tracking site ECI velocity vector (3 x 1), km/s
v_site_ijk = cross(earth_rotation_rate,r_site_ijk);

% Make sure the simulated observation is in
% the future with respect to the epoch
check_forward_motion = (JD - JD_epoch)*86400;

   if check_forward_motion >= 1

   % Write tracking site position vector to file.
   fprintf(fid6,'%18.8f %14.5f %15.5f %15.5f %15.5f\n',...
      JD,time,r_site_ijk(1),r_site_ijk(2),r_site_ijk(3));

   % Write satellite position and velocity truth to output file
   fprintf(fid9,'%18.8f %14.5f %14.5f %14.5f %14.5f %14.5f %14.5f %14.5f\n',...
      JD,time,X(1),X(2),X(3),X(4),X(5),X(6));

   % Satellite position and velocity vectors to pass
   % to subroutine 'razel' to generate data.
   rijk = [X(1); X(2); X(3)];
   vijk = [X(4); X(5); X(6)]; % Omitted comma before 'X(5)' term (after semi-colon)

   % Call to subroutine 'razel' to generate data
   [range,az,el,range_rate] ...
      = razel(rijk,vijk,elevation,lat,LST);
```

```
% Call to subroutine 'topocentric' to generate
% right ascension and declination data.
[range_topo,ra_topo,dec_topo,range_rate_topo,ra_topo_rate,dec_topo_rate]...
   = topocentric(rijk,vijk,r_site_ijk);

% Call to 'razel_ro' to generate psuedo-position state vector
% razel_ro is identical to razel, but set to return range as a vector
[range_sez,az_dummy,el_dummy,range_rate_dummy]...
   = razel_ro(rijk,vijk,elevation,lat,LST);

%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Apply some random error of magnitude equal to instrument sigmas
% for each observation type to the calculated data.
%
% NOTE -- Section between lines should be remarked-out for
% "perfect data" generation to do code validation
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%

range = range + randn * 0.20511; % approx 200 meter Don2radar range accuracy
az = az + randn * 0.0315; % approx .03 degrees radar accuracy az & el
el = el + randn * 0.0317; % approx .03 degrees radar accuracy az & el
% range_rate = range_rate + randn * 0.005;
% az_rate = az_rate + randn * 0.0002;
% el_rate = el_rate + randn * 0.0001;
% ra_topo = ra_topo + randn * 3.0;
% dec_topo = dec_topo + randn * 1.5;

% Pseudo-state position vector element errors based on pointing
% accuracy and geometry as discussed in thesis
% Values are for the Don-2N RADAR:
% Pseudo range error across track:
   range_sez(1) = range_sez(1) + randn*.20511;
% Pseudo range error along track
   range_sez(2) = range_sez(2) + randn*.20511;
% Pseudo range error above/below track
   range_sez(3) = range_sez(3) + randn*.20511;

% State vector errors for debugging filter performance
   X(1) = X(1) + randn * .1; %km
   X(2) = X(2) + randn * .1;
   X(3) = X(3) + randn * .1;
```

```
        X(4) = X(4) + randn * .0001;  %km/s
        X(5) = X(5) + randn * .0001;
        X(6) = X(6) + randn * .0001;

    % Ob_order: for example, the order of ob_type 1 is 2 since there
    % are two measurements (range and range-rate) per observation.
    fprintf(fid,'%1d %1d %24.16f %14.5f %12.5f %10.5f %10.5f %9.5f %10.5f %9.5f\n'...
        ,1,3,JD,time,range,az,el,lat,long,elevation);

    fprintf(fid2,'%1d %1d %24.16f %14.5f %14.5f %9.5f %9.5f %10.5f %9.5f\n'...
        ,2,2,JD,time,range,range_rate,lat,long,elevation);

    fprintf(fid3,'%1d %1d %24.16f %14.5f %10.5f %10.5f %9.5f %10.5f %9.5f\n'...
        ,3,2,JD,time,az,el,lat,long,elevation);

    fprintf(fid4,'%1d %1d %24.16f %14.5f %10.5f %10.5f %9.5f %10.5f %9.5f\n'...
        ,4,2,JD,time,ra_topo,dec_topo,lat,long,elevation);

    fprintf(fid5,'%1d %1d %24.16f %14.5f %8.5f %9.5f %10.5f %9.5f\n'...
        ,5,1,JD,time,range_rate,lat,long,elevation);

            % Write satellite position and velocity to output file
    fprintf(fid7,'%1d %1d %18.8f %14.5f %14.5f %14.5f %14.5f %14.5f %14.5f %14.5f
%9.5f %10.5f %9.5f\n'...
        ,7,6,JD,time,X(1),X(2),X(3),X(4),X(5),X(6),lat,long,elevation);

        % Write satellite position only to output file
    fprintf(fid8,'%1d %1d %18.8f %14.5f %14.5f %14.5f %14.5f %9.5f %10.5f %9.5f\n'...
        ,8,3,JD,time,range_sez(1),range_sez(2),range_sez(3),lat,long,elevation);

    end % end of 'check_forward_motion' logic check

    time = time + time_step

    % Update Julian Date
    JD = JD + time_step/86400.0

end % end of i = 1:num_points loop

% Close output data files.
fclose(fid);
fclose(fid2);
fclose(fid3);
fclose(fid4);
```

```
fclose(fid5);
fclose(fid6);
fclose(fid7);
fclose(fid8);
fclose(fid9);
status = 'Fin'

return

% end of file
```

Appendix I: Subroutine RAZEL MATLAB Code

```
function [range_sez_mag,az,el,range_rate] ...
   = razel(r_ijk,v_ijk,site_elevation,lat,LST)

% Capt Brian L. Foster
% 21 December 2002
% Edited by LtCol John Heslin
% 15 December 2004

% This is Algorithm 27 from Vallado (2001) pages 256.

% Earth radius in km
earth_rad = 6378.1363;

% Earth rotation rate, radians/s
%(this is a vector in the IJK coordinate system)
earth_rotation_rate = [0; 0; 0.000072921158553];

% Earth's shape eccentricity (squared) (no units)
earth_ecc_2 = 0.006694385;

% Convert input angles into radians
lat = lat * pi/180.0;
LST = LST * pi/180.0;

% Determine the tracking site's position vector.
% The auxiliary terms:

% C_earth (a.k.a. radius of curvature in the meridian), in km
C_earth = earth_rad/sqrt(1.0 - earth_ecc_2 * (sin(lat))^2);

% S_earth
S_earth = C_earth * (1.0 - earth_ecc_2);

% Horizontal component
r_del = (C_earth + site_elevation) * cos(lat);
% elevation is altitude above reference datum

% Vertical component
r_K = (S_earth + site_elevation) * sin(lat);
% elevation is altitude above reference datum

% Site vector in ECI (inertial) coordinate system
```

```
r_site_ijk = [r_del * cos(LST);
   r_del * sin(LST);
   r_K];

% Range vector in the IJK coordinate system, units in km
range_ijk = r_ijk - r_site_ijk;

% Tracking site inertial velocity, km/s
v_site_ijk = cross(earth_rotation_rate,r_site_ijk);

% Units km/s
relative_velocity_ijk = v_ijk - v_site_ijk;

% Rotation matrix from IJK to SEZ coordinate system
IJK_TO_SEZ = [sin(lat)*cos(LST) sin(lat)*sin(LST) -cos(lat);
   -sin(LST) cos(LST) 0;
   cos(lat)*cos(LST) cos(lat)*sin(LST) sin(lat)];

% Rotate IJK vector to SEZ
% Range
range_sez = IJK_TO_SEZ * range_ijk;

% Relative velocity, SEZ
relative_velocity_sez = IJK_TO_SEZ * relative_velocity_ijk;


% Range magnitude, in kilometers
range_sez_mag = norm(range_sez);

% Elevation, in degrees (Vallado definition)
%el = (asin(range_sez(3)/range_sez_mag)) * 180.0/pi;

% Intermediate variable since it occurs several times
rhos2rhoe2 = range_sez(1)^2 + range_sez(2)^2;


% Elevation, in degrees (Wiesel definition)
el = atan(range_sez(3)/sqrt(rhos2rhoe2))*180.0/pi;

% Azimuth, in degrees (Wiesel definition)
fid1 = fopen('range_az_el_residuals_output.txt','w+');
fprintf(fid1,'%14.8f %14.8f %14.8f\n',...
      range_sez(1),range_sez(2),range_sez(3));
```

```
% Azimuth
if(el ~= 90.0)
   sin_az = range_sez(2)/sqrt(rhos2rhoe2);

   cos_az = -range_sez(1)/sqrt(rhos2rhoe2);
end

% Azimuth, in degrees
az = atan2(sin_az,cos_az) * 180.0/pi;

if(az < 0.0)
   az = az + 360.0;
end

if(az > 360.0)
   az = az - 360.0;
end

% Range rate magnitude
% Vallado page 174 is unclear whether the range/range rate
% used next is IJK or SEZ.  I assume SEZ.  Units in km/s.
range_rate = dot(range_sez,relative_velocity_sez)/range_sez_mag;
fclose(fid1);
return
```

Appendix J: Subroutine LSTime MATLAB Code

```
function [GST,LST] = lstime(JD,long)

% function - calculates the local sidereal time for a given longitude.
% Capt Brian L. Foster
% 21 December 2002
% Edited by LtCol John Heslin
% 15 December 2004

% This is Algorithm 15 from Vallado (2001) page 192.
% It calculates the local sidereal time for a given longitude.

format long g

% Inputs:
% JD_not = Julian Date at beginning of the day of interest (0 h UT1)
% UT1 = Universal time of interest (elapsed time since midnight in seconds)
% Longitude = longitude of tracking (sensor) site
%   East longitudes: 0.0 to 180.0 are positive
%   West longitudes: 0.0 to -180.0 are negative

% Take the integer portion of the Julian Date.
% This equal 1200 hours UTC on the day of interest.
int_day = floor(JD);

% Find the fraction of the day.
fract_day = mod(JD,int_day);

% Determine if the fractional part of the day
% is before or after midnight (OOOO hours UTC).

if (fract_day > 0.5)
   JD_not = JD - fract_day + 0.5;
else
   JD_not = JD - fract_day - 0.5;
end

% Now calculate UT1, elapsed time since midnight in seconds.
% 86400.0 is seconds/day.
if (fract_day > 0.5)
   UT1 = (fract_day -0.5)*86400.0;
else
   UT1 = (fract_day + 0.5)*86400.0;
```

```
end

% 2,451,545.0 is the Julian Date for the epoch of J2000
% (1 January 2000 12:00).
T_ut1 = (JD_not - 2451545.0)/36525;

%%%%%%%%%%%%%%%%%%%
% Greenwich Mean Sidereal Time (GST_not)
% at midnight (0 h, 0 min, 0 sec)UT1 degrees (Vallado, page 60)
% GST_not in degrees
GST_not = 100.4606184 + (36000.77005361 * T_ut1)...
   + (0.00038793 * T_ut1^2) - (2.6 * 10^-8)* T_ut1^3;
%%%%%%%%%%%%%%%%%%%

% Greenwich Sidereal Time (GST) at the time of interest
% (observation time).
% GST = GST_not + earth rotation rate x
% elapsed time since midnight.

% GST in degrees
GST = GST_not + 0.2506844733746215/60.0 * UT1;

% Reduce GST_deg to values less than 360 degrees,
% either positive or negative (convert to positive).
if(GST > 360.0)
   GST = mod(GST,360.0);
end

if(GST < -360.0)
   GST = mod(GST,-360.0);
end

%%%%%%%%%%%%%%%%%%%

% Local Sidereal Time (LST) at the longitude of interest
LST = GST + long;

% Reduce LST_deg to values less than 360 degrees,
% either positive or negative (convert to positive).
if(LST > 360.0)
   LST = mod(LST,360.0);
end

if(LST < -360.0)
```

```
   LST = mod(LST,-360.0);
end

return
```

Appendix K: Subroutine Gibbs Vectors MATLAB Code

```
% Capt Brian L. Foster
% 21 December 2002
% Edited by LtCol John Heslin
% 15 December 2004

% Output is Gibbs vectors from site - reads 'range_az_el_data' file
load('range_az_el_data.txt','-ascii') % created by data generator
[ob_type,order_ob,JD,ob_time,range,az,el,lat,long,alt]...
   =textread('range_az_el_data.txt','%d %d %f %f %f %f %f %f %f %f',-1);

JD
fid1 = fopen('Gibbs_vectors_output.txt','w+');
% can create a different output for different sites by changing "data
% generator" site info and changing (say the end of) this output file name
% to identify the different site

size_JD = length(JD)
  size_ob_time = size(ob_time)

data_vector_size = size(range);

num_pts =data_vector_size(1)

for i = 1:num_pts

 i

 JDAY = JD(i);
  longitude = long(i);
  altitude = alt(i);
  latitude = lat(i);

  range_rate = 0.0;
  az_rate = 0.0;
  el_rate = 0.0;

  % 2. subroutine 'lstime' to get Local Sidereal Time
       %   for the tracking site.  (Greenwich Sidereal Time, GST,
       %   not used).  LST in degrees.

  [GST,LST] = lstime(JDAY,longitude)
```

```
    % subroutine site_track returns Site vector
        %   in ECI (inertial) coordinate system,
        %   Satellite absolute position and velocity in the ECI (inertial)
        %   IJK coordinate system. (r_ijk, v_ijk) Units in km and km/s.

    [r_ijk,v_ijk,r_site_ijk] =...
    site_track(latitude,LST,altitude,range(i),az(i),el(i), range_rate, az_rate,el_rate);

r_ijk(1);
r_ijk(2);
r_ijk(3);

x = r_ijk(1);
    y = r_ijk(2);
    z = r_ijk(3);

    fprintf(fid1,'%24.16f %14.5f %14.5f %14.5f\n'...
    ,JDAY,x,y,z);

end

fclose(fid1)

% end of file
```

Appendix L: Subroutine Gibbs MATLAB Code

```
function [v2,warning] = gibbs(r1,r2,r3)

% Determines orbit (gives v2) from 3 pos'n vectors (far)& times

% Capt Brian L. Foster
% 23 December 2002
% Edited by LtCol John Heslin
% 15 December 2004

% This is Algorithm 51 from Vallado (2001) page 440.
% It returns the velocity vector associated with position
% vector r2.

% The input vectors r1, r2, and r3 are in the IJK coordinate system
% and with units of kilometers.

% Test -- China observation vectors for sts-109:
% r1=[-5822.31523    -2180.56929     2928.64129]';
% r2=[-5441.86853    -2859.79024     3080.24747]';
% r3=[-4985.26339    -3508.55170     3191.58315]';

% Result from China data:
% v2 =
%
%          4.03070849653616
%         -6.40200313948651
%          1.26925203316625
% Therefore the initial state vector for the lsq estimator will be
% x1= -5441.86853
% x2=  -2859.79024
% x3= 3080.24747
% x4= 4.03070849653616
% x5= -6.40200313948651
% x6= 1.26925203316625
% with JD = 2452340.3338426971
% Note -- The middle vector is in the middle of the data run, so the epoch
% would start in the middle of the data run.  In practical use, choose
% vectors close to the start of the data run to use as a reference state


format long g
```

```
% Earth's gravitational parameter, km^3/s^2
mu = 398600.4415;

% Normal vectors
Z12_vec = cross(r1,r2);
Z23_vec = cross(r2,r3);
Z31_vec = cross(r3,r1);

% Vectors are coplanar if Z23_vec is perpendicular to r1.

% Magnitudes of the position vectors
r1_mag = norm(r1);
r2_mag = norm(r2);
r3_mag = norm(r3);

% Check to see how coplanar the vectors are.
alpha_cop = 90.0 - acos(dot(Z23_vec,r1)/...
   (norm(Z23_vec)*r1_mag))*180.0/pi

% Determine angular separations to ensure sufficient separation
% Angular separation between r1 and r2, in degrees
alpha12 = acos(dot(r1,r2)/(r1_mag*r2_mag))*180.0/pi

% Angular separation between r2 and r3, in degrees
alpha23 = acos(dot(r2,r3)/(r2_mag*r3_mag))*180.0/pi

if(alpha12 < 1.0 | alpha23 < 1.0)
   warning = 'r1, r2, and r3 are too close.  Use Herrick-Gibbs.'
   v2 = 'v2 not calculated.'
   return
end

% Intermediate vectors
N_vec = r1_mag * Z23_vec + r2_mag * Z31_vec + r3_mag * Z12_vec;

D_vec = Z12_vec + Z23_vec + Z31_vec;

S_vec = (r2_mag - r3_mag)*r1 + ...
   (r3_mag - r1_mag)*r2 + (r1_mag - r2_mag)*r3;

B_vec = cross(D_vec,r2);

Lg = sqrt(mu/(norm(N_vec) * norm(D_vec)));
```

```
% Velocity vector associated with r2, units in km/s
v2 = Lg/r2_mag * B_vec + Lg * S_vec
warning = 0;
return
```

Appendix M: Subroutine H-Gibbs MATLAB Code

```
function [v2] = h_gibbs(r1,r2,r3,JD1,JD2,JD3)

% function determines orbit (v2) given three position vectors (close) and times

% Capt Brian L. Foster
% 23 December 2002
% Edited by Lt Col Heslin
% 15 December 2004

% This is Algorithm 52 from Vallado (2001) page 446.

% Test case vectors
r1 = [1607.879850;-7026.697450; -15.031650]
r2 = [1599.998580; -7028.257130; 14.877410]
r3 = [1592.705670; -7030.083050; 44.770310]

% Julian Dates of test case vectors
JD1 = 2452734.4999537
JD2 = 2452734.5
JD3 = 2452734.5000463

format long g

% Earth's gravitational parameter, km^3/s^2
mu = 398600.4415;

% The position vectors r1, r2, and r3 are in the IJK
% coordinate system with units of kilometers.

% Remember that JD dates are in "DAYS" and must be
% converted to seconds.

del_t31 = (JD3 - JD1)*86400.0;
del_t32 = (JD3 - JD2)*86400.0;
del_t21 = (JD2 - JD1)*86400.0;

% Data for test case debugging.
% del_t31 = 153.04;
% del_t32 = 76.56;
% del_t21 = 76.48;

Z23_vec = cross(r2,r3);
```

```
Z23 = norm(Z23_vec);

r1_mag = norm(r1);
r2_mag = norm(r2);
r3_mag = norm(r3);

alpha_cop = 90.0 - acos(dot(Z23_vec,r1)/(Z23*r1_mag))*180.0/pi

% Determine angular separations to ensure sufficient separation
% Angular separation between r1 and r2
alpha12 = acos(dot(r1,r2)/(r1_mag*r2_mag))*180.0/pi

% Angular separation between r2 and r3
alpha23 = acos(dot(r2,r3)/(r2_mag*r3_mag))*180.0/pi

if(alpha12 > 5.0 | alpha23 > 5.0)
   v2 = 'v2 not calculated.';
   warning = 'r1, r2, and r3 are too far apart.  Use Gibbs method.';
   return
end

% Velocity vector associated with second position vector in km/s.
v2 = -del_t32*(1/(del_t21*del_t31) + mu/(12*r1_mag^3))*r1 +...
   (del_t32 - del_t21)*(1/(del_t21*del_t32) + mu/(12*r2_mag^3))*r2 +...
   del_t21*(1/(del_t32*del_t31) + mu/(12*r3_mag^3))*r3
warning = 0;
return
```

Appendix N: Obtaining the Satellite Orbital Analysis Program (SOAP)

(*reprinted from SOAP Users Manual*)

Distribution of SOAP is restricted to U.S. Government personnel and U.S. corporations with a current Government contract involving space systems. Other organizations may request SOAP, but such requests will require special approval from Aerospace and U.S. Space and Missile Systems Center. The distribution procedure varies based on the organizational affiliation of the individual requesting the software. The Aerospace point of contact is:

Sharon Robinson
Mail Stop M1/039
The Aerospace Corporation
P.O. Box 92957
Los Angeles CA 90009-2957
Voice: (310) 336-0384
Fax: (310) 336-5706
E-mail: Sharon.J.Robinson@aero.org

Users may be asked to draft a written request using their organization's letterhead. When a written request is received, Aerospace will send a license agreement will be sent for the user to sign and return. The software will be sent after this process is completed. Processing can take up to 30 days.

Bibliography

[1] Cho, Sangwoo and Joohwan Chun. "Bayesian Bootstrap filtering for Multiple Mobile Position Determination using LEO Satellites." IEEE Vehicular Technology Conference, 2000, Vol. 2. pp1920-1924.

[2] Chobotov, Vladimir A. "Orbital Mechanics (3rd Edition)." Reston VA: American Institute of Aeronautics and Astronautics, Inc., 2002.

[3] Foster, Brian L. "Orbit Determination for a Microsatellite Rendezvous with a Noncooperative Target." MS thesis, AFIT/GAI/ENY/03-2. Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base OH, March 2003.

[4] Hoffman, Donald and Richard Hujsak. "Real Time Orbit Determination with Sequential Filters." ADPA/NSIA Satellite Command and Control Conference, Reston, VA: Sept 1997.

[5] Hoots, Felix R. "Measurement Bias Effects on Satellite Estimation and Prediction (AAS-01-337)." Conference Proceedings: Astrodynamics; Advances in the Astronautical Sciences, Vol. 109, Pt. 1. 2001, pp 529-544.

[6] Kelso, T.S., "NORAD Two-Line Element Sets Historical Archives, US Space Shuttle." N. pag. http://www.celestrak.com/NORAD/archives/STS/sts-109.txt. 4 January 2005

[7] Kim, Soohong and Joohwan Chun. "Bayesian Bootstrap Filtering for the LEO Satellite Orbit Determination." IEEE Vehicular Technology Conference, 2000, Vol. 2. pp1611-1615.

[8] Rumford, Timothy E. "Demonstration of Autonomous Rendezvous Technology (DART) Project Summary." Orbital Sciences Corp., VA. 2003

[9] Tansey, R.J., B. Campbell, and A. Koumvakalis. "Description and Experimental Results of a 58-lb Portable LEO Satellite Tracker," *Proceedings of SPIE,* Vol. 3434 (1998), pp. 78-87.

[10] Toso, Alan R., "Systems-Level Feasibility Analysis Of A Microsatellite Rendezvous With Non-Cooperative Targets." MS thesis, AFIT/GSS/ENY/04-M06. Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base OH, March 2004.

[11] "The Near Intersection of Two Skew Lines" http://sun.uni-regensburg.de/idl-5.5/html/idl4/jhuapl.doc/vectors/v_skew.html.

[12] Tragesser, Steven G. Class handouts, MECH 532, Introduction to Space Flight Mechanics; MECH 533, Intermediate Space Flight Dynamics. Graduate School of Engineering and Management, Air Force Institute of Technology, Wright-Patterson AFB OH, January 2003.

[13] Truszkowski, Walt. "An Agent Community Supporting Orbit Determination." Proceedings, Florida Artificial Intelligence Research (FLAIRS) Conference. American Association for Artificial Intelligence (AAAI Press), Menlo Park, CA: 2002.

[14] Tschirhart, Troy A. "A Study of Control Laws for Microsatellite Rendezvous with a Noncooperative Target." MS Thesis, AFIT/GAI/ENY/03-3. Graduate School of Engineering and Management, Air Force Institute of Technology (AU), Wright-Patterson Air Force Base OH, March 2003.

[15] Vallado, David A. "Fundamentals of Astrodynamics and Applications (2nd Edition)." El Segundo CA: Microcosm Press, 2001.

[16] Vallado, David A. and Scott S. Carter. "Accurate Orbit Determination From Short-Arc Observational Data (AAS-97-704)." Advances in the Astronautical Sciences, Vol. 97, No. 2. 1998, pp 1587-1601.

[17] Wertz, James R. and Wiley J. Larson. "Space Mission Analysis and Design (3rd Edition)." El Segundo CA: Microcosm Press, 1999.

[18] Wiesel, William E. "Modern Orbit Determination." Beavercreek, OH: Aphelion press, 2003.

Vita

Lieutenant Colonel John P. Heslin graduated from the Virginia Military Institute with a Bachelor of Science degree in Mechanical Engineering and a commission in the USAF in May 1988. His first assignment was to Wright-Patterson AFB, Ohio, where he worked as an engineer and cross-trained as an intelligence officer. In May 1994, while stationed at Wright-Paterson AFB, he graduated from the Joint Military Intelligence College with a Master of Science in Strategic Intelligence.

Lieutenant Colonel Heslin was next assigned to the 613th Air Intelligence Flight on Anderson AFB, Guam as the Chief of Intelligence Operations, a post that required frequent travel to Thailand, the Philippines, Indonesia, and Singapore, to meet and brief each respective nation's Chief of Air Force Intelligence. In April 1997, he was assigned to the 36th Fighter Squadron, Osan AB, South Korea as the squadron intelligence officer. His next assignment began in April 1998, as the Wing Intelligence Officer for the 100th Air Refueling Wing, RAF Mildenhall, England. While with the 100th, shortly following his success as the Chief of Tanker Intelligence during Operation Allied Force, he became the 100th Operations Support Squadron Assistant Operations Officer, leading squadron flight commanders as well as overseeing the flying schedule and aircrew training.

He was assigned to the Directorate of Space Operations and Integration, Headquarters Air Force, Pentagon, in May 2001, as the Chief, Ground-Based Warning Systems Program Element Monitor, responsible for 17 Integrated Tactical Warning and Attack Assessment programs worth over $4Billion. In June, 2001, he graduated from the

Touro University correspondence program with a Master of Business Administration. In response to the September 11, 2001 attack on the Pentagon, he was assigned to additional duties as the USAF Crisis Action Team Executive Officer.

In September 2003, he entered the Graduate School of Engineering and Management, Air Force Institute of Technology. Upon graduation, he will be assigned to the Defense Intelligence Agency, Field Operating Base, Korea.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 074-0188

The public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of the collection of information, including suggestions for reducing this burden to Department of Defense, Washington Headquarters Services, Directorate for Information Operations and Reports (0704-0188), 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to an penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.
PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ADDRESS.

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From – To)*
March 2005	Master's Thesis	March 2004 – March 2005

4. TITLE AND SUBTITLE
ORBIT ESTIMATION ALGORITHMS FOR A MICROSATELLITE RENDEZVOUS WITH A NON-COOPERATIVE TARGET

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Heslin, John P., Lt Col, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAMES(S) AND ADDRESS(S)
Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way, Building 640
WPAFB OH 45433-8865

8. PERFORMING ORGANIZATION REPORT NUMBER
AFIT/GSS/ENY/05-M02

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This study investigated the minimum requirements to establish a satellite tracking system architecture for a microsatellite to rendezvous with a non-cooperative target satellite. A prototype optical tracking system was reviewed with emphasis on a proposed tactical employment that could be used by technologically unsophisticated state or non-state adversaries. With the tracking system architecture selected, simulated tracking data was processed with a Non-Linear Least Squares batch orbit estimation algorithm and a Bayes sequential orbit determination filter to update the target satellite's state vector.

15. SUBJECT TERMS
Microsatellite, Orbit Determination, Non-linear Least Squares, Bayes

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			Dr. Richard G. Cobb, ENY
U	**U**	**U**	**UU**	184	**19b. TELEPHONE NUMBER** *(Include area code)* (937) 255-6565, ext 6378 (richard.cobb@afit.edu)

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std. Z39-18